

GRAPH CUTS, SUM-OF-SUBMODULAR FLOW, AND LINEAR PROGRAMMING: EFFECTIVE INFERENCE IN HIGHER-ORDER MARKOV RANDOM FIELDS

A Dissertation

Presented to the Faculty of the Graduate School
of Cornell University

in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by

Alexander Fix

May 2017

© 2017 Alexander Fix
ALL RIGHTS RESERVED

GRAPH CUTS, SUM-OF-SUBMODULAR FLOW, AND LINEAR
PROGRAMMING: EFFECTIVE INFERENCE IN HIGHER-ORDER MARKOV
RANDOM FIELDS

Alexander Fix, Ph.D.

Cornell University 2017

Optimization algorithms have a long history of success in computer vision, providing effective algorithms for tasks as varied as segmentation, stereo estimation, image denoising and scene understanding. A notable example of this is Graph Cuts, in which the minimum-cut problem is used to solve a class of vision problems known as first-order Markov Random Fields. Despite this success, first-order MRFs have their limitations. They cannot encode correlations between groups of pixels larger than two or easily express higher-order statistics of images. In this thesis, we generalize graph cuts to higher-order MRFs, while still maintaining the properties that make graph cuts successful.

In particular, we will examine three different mathematical techniques which have combined to make previously intractable higher-order inference problems become practical within the last few years. First, order-reducing reductions, which transform higher-order problems into familiar first-order MRFs. Second, a generalization of the min-cut problem to hypergraphs, called Sum-of-Submodular optimization. And finally linear programming relaxations based on the Local Marginal Polytope, which together with Sum-of-Submodular flow results in the highly effective primal-dual algorithm SoSPD.

This thesis presents all mathematical background for these algorithms, as well as an implementation and experimental comparison with state-of-the-art.

BIOGRAPHICAL SKETCH

Alexander Fix graduated with a Bachelor of Science in Computer Science and Mathematics from the University of Chicago in 2009. He is currently a PhD student at Cornell University, focusing on optimization algorithms with applications in Computer Vision. His advisor is Ramin Zabih. In the summer of 2013, he was a research intern at Google, advised by Sameer Agarwal. From 2013 to 2014, he completed his PhD research at Cornell Tech, in NYC. Since February 2015, he has been a researcher at Oculus Research in Redmond, WA.

ACKNOWLEDGEMENTS

First, I am most grateful to my PhD advisor, Ramin Zabih, who has supported me without fail for the last six years. There are too many things to list, but here's a start: Thank you for giving me the perfect environment to grow as a researcher — for independence when I needed it, and guidance when independence failed. Thank you for your example of how to be a member of a research community, and all your many introductions — here's to many more workshops in Italy. Thank you for bearing with me on the actual writing of this thesis — it's been a trial, but I think it's turned out in the end.

I would also like to thank my committee members David Williamson and David Shmoys, for teaching me everything I know about approximation algorithms and linear programming, and for all your questions along the way — this thesis wouldn't be half so interesting without them. Endre Boros, without whom I would not have started on my first project, and for being a continual fount of promising ideas and research ideas ever since. Sameer Agarwal, and the rest of Steve Seitz's group at Google, for a truly wonderful internship — thank you for introducing me to the wonderful world of research in industry.

And finally, Lorraine, for putting up with all the years, and all the travel. I couldn't have done it without you.

This research in this thesis has been funded by NSF grants IIS-0803705, IIS-1161860/1161476, and IIS-1161282.

TABLE OF CONTENTS

Biographical Sketch	iii
Acknowledgements	iv
Table of Contents	v
1 Introduction	1
1.1 Notation	3
1.2 Optimization Basics	4
1.2.1 Constrained Optimization	6
1.2.2 Constraint Indicator Functions	7
1.2.3 Minimizing Elements	8
1.2.4 Relaxations	9
1.2.5 Equivalence of Optimization Problems	11
1.2.6 Common Equivalences Between Problems	13
1.3 Example: Image Segmentation	14
1.3.1 Binary Labeling Problems	15
1.3.2 Per-Pixel Cost Functions	15
1.3.3 Spatial Relations Between Pixels	17
1.3.4 The Potts Model	18
1.3.5 Reduction to Graph Cut	19
1.3.6 Discussion	22
1.4 Markov Random Fields	23
1.4.1 Labeling Problems	23
1.4.2 Maximum A-Posteriori (MAP) Inference	24
1.4.3 Log-probabilities	26
1.4.4 MAP inference in Foreground-Background Segmentation	27
1.4.5 Conditional Dependence	29
1.4.6 The Hammersley-Clifford Theorem	31
1.4.7 The Potts Model as an MRF	33
1.5 First-order and Higher-order MRFs	34
1.5.1 Advantages of Higher-Order Models	35
1.5.2 Image Denoising and Patch-Based Priors	36
1.5.3 Curvature Regularizing Priors for Stereo	40
1.6 Conclusion	43
2 Mathematical Background	45
2.1 Reparameterization	46
2.2 Pseudoboolean functions	48
2.2.1 Representations of MRFs	49
2.2.2 Set Functions	50
2.2.3 Multilinear Polynomials	51
2.2.4 Properties of Multilinear Polynomials	51

2.2.5	Computational Complexity and Hardness of Approximation	53
2.3	Submodular Functions	58
2.3.1	Decreasing Marginal Gains	58
2.3.2	Equivalent Definitions of Submodularity	60
2.3.3	Properties of Submodular Functions	62
2.3.4	Submodular First-order Pseudoboolean Functions	65
2.4	Local and Marginal Polytopes for MRFs	67
2.4.1	Weighted Averages as Linear Programs	68
2.4.2	Marginal polytopes	69
2.5	Linear Programming	72
2.5.1	Linear Cone Programming	75
2.6	Convex Sets and Convex Functions	76
2.7	Duality	78
2.7.1	Exchanging Minimization and Maximization	79
2.7.2	Linear Programming Duality: An Example	80
2.7.3	Conic Duality	86
2.8	Optimality for Linear Programs	90
2.9	Duality for the Local Marginal Polytope	92
2.10	First-order Binary MRFs and Minimum Cut	94
2.10.1	Solving First-order Submodular MRFs with Graph Cuts	95
2.10.2	Linear Programs for Min-Cut	97
2.10.3	Local Marginal Polytope for First-order Binary Problems	98
3	Related Work	101
3.1	Higher-Order Models in Computer Vision	102
3.2	Inference Algorithms for Binary MRFs	104
3.2.1	First-Order Submodular MRFs	104
3.2.2	First-Order Nonsubmodular MRFs	105
3.2.3	Higher-Order Reductions	107
3.2.4	Higher-order Submodular Functions	109
3.3	Primal Algorithms	111
3.4	Dual Algorithms	114
3.5	Primal-Dual Algorithms	116
4	Higher order reductions	118
4.1	Introduction	119
4.2	Related work	120
4.2.1	Reduction by substitution	121
4.2.2	Reducing negative-coefficient terms	122
4.2.3	Reducing positive-coefficient terms	122
4.2.4	Generalized Roof Duality	123
4.3	Reducing groups of higher-order terms	125
4.3.1	Our method	127

4.4	Worst case performance	129
4.5	Local completeness	130
4.5.1	Performance on locally complete problems	131
4.6	Locally complete energy functions in vision	133
4.7	Experimental results	134
5	Sum of Submodular Minimization	140
5.1	Sum of Submodular Minimization via Submodular Flow	141
5.1.1	Definitions and Graph Construction	142
5.1.2	Flow as a Reparameterization	143
5.1.3	The Max-Flow Min-Cut Theorem for SoS Functions	145
5.2	IBFS for Submodular Flow	148
5.2.1	IBFS on Graphs	149
5.2.2	Modifying IBFS for SoS Flow	150
5.2.3	Running Time	152
5.3	Proof of the “No Shortcuts” Lemma	152
5.4	The Current Arc Heuristic	155
6	Submodular Upper Bounds for Higher Order Energy Functions	159
6.1	Introduction	159
6.2	Background and Related Work	160
6.2.1	Notation	162
6.3	Submodular Upper Bounds	163
6.4	Upper Bound Approximations	164
6.4.1	The Iterative Heuristic of SoSPD	165
6.4.2	Quadratic-Based Submodular Upper Bounds	166
6.4.3	Cardinality-Based Submodular Upper Bounds	169
7	A Primal-Dual Algorithm for Higher-Order Multilabel Markov Random Fields	175
7.1	Higher-order Multi-label MRFs	175
7.1.1	Summary of Our Method	176
7.2	Related Work	178
7.2.1	Graph Cut Methods and Higher-Order MRFs	178
7.2.2	Linear Programming and Duality for MRFs	178
7.2.3	Sum-of-Submodular Flow	179
7.3	The SoS Primal Dual Algorithm	181
7.3.1	Update-Duals-Primals	184
7.3.2	Pre-Edit-Duals	186
7.3.3	Post-Edit-Duals	187
7.3.4	Proof of Convergence	188
7.3.5	Approximation Bounds	189

8	Experimental Evaluation of the SoSPD Algorithm	192
8.1	Benchmarks and Datasets	192
8.1.1	Field of Experts Denoising	193
8.1.2	Curvature Regularizing Stereo Reconstruction	194
8.2	Comparison of Upper Bound Methods	195
8.2.1	Experimental Setup	195
8.2.2	Results	198
8.3	Evaluation of SoSPD	201
8.3.1	Stereo reconstruction	203
8.3.2	Field of Experts denoising	206
9	Structured learning of sum-of-submodular higher order energy functions	207
9.1	Introduction	207
9.2	Related Work	209
9.3	S3SVM: SoS Structured SVMs	210
9.3.1	Structured SVMs	211
9.3.2	Submodular Feature Encoding	212
9.3.3	Solving the quadratic program	214
9.3.4	Generalization to multi-label prediction	216
9.4	Experimental Results	218
9.4.1	Binary denoising	219
9.4.2	Interactive segmentation	220
10	Conclusion	224
A	Local Completeness	227
B	Laplacian Equations	231
C	Approximation Ratio for Cardinality Upper Bounds	233
	Bibliography	235

CHAPTER 1

INTRODUCTION

Optimization algorithms have a long history of success in computer vision, providing the basis for many effective tasks as varied as segmentation, stereo estimation, image denoising and scene understanding. A particularly notable example of this is the method of Graph Cuts [11], in which minimum-cut algorithms are used to solve a class of vision problems known as first-order Markov Random Fields (MRFs). There are two main reasons for Graph Cuts' success. First, min-cut is already a well-studied problem with highly efficient algorithms (and the popularity of Graph Cuts has encouraged the development of even more efficient algorithms tuned specifically to computer vision problems). Second, the class of problems solved by Graph Cuts (first-order MRFs) encapsulates the fundamental idea of image locality, i.e., that pixels in an image are highly correlated with their neighbors. This property makes MRFs well-suited to solving a wide range of inference problems in computer vision as well as machine learning and other fields.

Despite this success, first-order MRFs have their limitations. They cannot easily encode correlations between groups of pixels larger than two, and thus are unable to express higher-order statistics of images. In this thesis, we focus on removing this limitation. Our goal is to generalize graph cuts to a wider class of higher-order MRFs, greatly extending the class of models for which MRF inference can be applied, while keeping the fast algorithms that make graph cuts successful. In a broader sense, this thesis is about the interaction between modeling and inference: by applying new advances in algorithms, we can now optimize a new class of models which were previously intractable, allowing

much greater flexibility and power in the kinds of problems we can solve.

Our goal for the first two chapters is to cover the mathematical background for Markov Random Fields, and to introduce the main optimization problem considered in the later chapters, which is a minimization problem of the form:

$$\min_{\mathbf{x}} \sum_i f_i(x_i) + \sum_C f_C(\mathbf{x}_C) \quad (1.1)$$

where the vector of variables \mathbf{x} comes from a discrete label space $\mathbf{x} \in \prod_i \mathcal{X}_i$, and the function f to be optimized is a sum of unary functions f_i (each depending on a single variable x_i) and so-called clique functions f_C each of which depends on a subset of the variables x_C from a clique C , which are overlapping subsets of the variables.

In particular, the main results of this thesis rely on the concepts of Linear Programming, duality, and linear programming relaxations. This introduction and the following chapter discuss the necessary background for these topics. We begin with the basic concepts of optimization, which may be unnecessary for readers familiar with the subject. However, we wish to put the MRF inference problem in the context of probabilistic inference, which informs the types of models which are useful in computer vision.

In this chapter, we will give a brief introduction to the use of optimization algorithms in computer vision, along with an extended example of how first order MRFs and graph cuts are applied to a simple but typical vision task of binary segmentation. We will also explain how first order MRFs can be generalized to include interactions between more than just pairs of pixels — such MRFs are called higher-order. We will conclude with several applications where allowing these higher-order interactions is necessary for using more sophisticated models which cannot be expressed by simpler first-order MRFs.

1.1 Notation

All special notation will be introduced at the point of first use in the text, but is also repeated here for easy reference.

We will write vectors as bold lowercase symbols: \mathbf{x} . For any sets \mathcal{X} and S , \mathcal{X}^S is the set of all vectors with components in \mathcal{X} indexed by the elements of S . This allows, for instance, vectors not indexed by just the set $1, \dots, n$. We'll use \mathcal{X}^n as shorthand for $\mathcal{X}^{\{1, \dots, n\}}$.

For $i \in S$ and $\mathbf{x} \in \mathcal{X}^S$, x_i is the i -th component of \mathbf{x} . For any subset $T \subseteq S$ and vector $\mathbf{x} \in \mathcal{X}^S$, we'll write \mathbf{x}_T for the subvector of S corresponding to just the components in T .

We will always use V for the set of variable indices, so that x_i are indexed by $i \in V$. When summing over variables, we will write \sum_i as shorthand for $\sum_{i \in V}$, as in (1.1). Similarly, we'll use C to denote the set of cliques, and will use \sum_C as shorthand for $\sum_{C \in \mathcal{C}}$.

When the cliques are all pairs, $|C| = 2$, then we have a graph with unordered edges $\{i, j\}$. We'll write pairwise clique functions as $f_{i,j}$ which are similarly unordered, i.e., $f_{i,j}(x_i, x_j) = f_{j,i}(x_j, x_i)$. Sums over pairs are also unordered, so $\sum_{i,j}$ means $\sum_{i < j}$, including each unordered pair only once. In a graph, we will use $N(i)$ to denote the set of neighbors of i , $N(i) = \{j \mid \{i, j\} \in E\}$.

The minimum value of a minimization problem (or maximum value of a maximization problem) is denoted OPT , so the minimum value of (1.1) is $\text{OPT}(1.1)$. Minimizers are denoted by \mathbf{x}^* , and X^* is the set of all minimizers.

For a finite set X , the set of probability distributions on X is $\mathcal{P}[X]$, i.e., the set of all $p : X \rightarrow \mathbb{R}$ with $p(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in X$, and $\sum_{\mathbf{x} \in X} p(\mathbf{x}) = 1$. For any function $f : X \rightarrow \mathbb{R}$, the expectation of f under the probability distribution p is $\langle f, p \rangle$, which is given by $\langle f, p \rangle := \sum_{\mathbf{x} \in X} f(\mathbf{x})p(\mathbf{x})$. Note that this is the inner product of f and p when treated as vectors in \mathbb{R}^X , so we will use $\langle \cdot, \cdot \rangle$ for inner products in general.

The normal distribution with mean μ and standard deviation σ is $N(\mu, \sigma)$. We write $x \sim N(\mu, \sigma)$ to denote a random variable drawn from this distribution. We use \propto to denote proportionality, so the probability distribution function of $N(\mu, \sigma)$ is $p(x) \propto e^{-\frac{(x-\mu)^2}{2\sigma^2}}$.

The Iverson bracket $\llbracket P(x) \rrbracket$ is 1 or 0 depending on whether the condition $P(x)$ is true or false. So $f(x) = \llbracket x \text{ is even} \rrbracket$ has $f(3) = 0$ and $f(6) = 1$.

1.2 Optimization Basics

Optimization, at its core, is a search problem — we have an exponentially large (or possibly infinite) set of choices, among which we want to find the “best” one. To be precise, the most general formulation of optimization is that we have a *solution space* \mathcal{X} (also called a *state space* or *feasible set*) and some *objective* (or *cost function*) function $f : \mathcal{X} \rightarrow \mathbb{R}$, saying how good a given solution is. Our goal is to find an $\mathbf{x} \in \mathcal{X}$ which minimizes the objective value $f(\mathbf{x})$.

Our standard notation for an optimization problem is:

$$\begin{aligned} \min_{\mathbf{x}} f(\mathbf{x}) \\ \text{s.t. } \mathbf{x} \in \mathcal{X} \end{aligned} \tag{1.2}$$

For small problems which fit on one line, we will also write

$$\min_{\mathbf{x}} \{f(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\}. \quad (1.3)$$

Most of the optimization problems we'll consider are minimization problems, where the goal is to find an \mathbf{x} with $f(\mathbf{x})$ as small as possible. We will write the optimum value (either minimum or maximum) of an optimization problem as $\text{OPT}(1.2)$. Maximization problems will arise later, particularly when we come to the topic of *duality*. Note that we can convert back and forth between maximization and minimization problems by using the identity

$$\max_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) = - \min_{\mathbf{x} \in \mathcal{X}} -f(\mathbf{x}). \quad (1.4)$$

By itself, having an objective function isn't much use — if we know nothing at all about the function f (i.e., we have a black-box which given an $\mathbf{x} \in \mathcal{X}$, evaluates and returns $f(\mathbf{x})$) then this general optimization problem is as hard as a totally unguided search problem — the best possible algorithm is to evaluate every single $f(\mathbf{x})$ and return the best one. Since the set \mathcal{X} is exponentially (or infinitely) large for most interesting problems, this tells us there cannot be an efficient optimization algorithm that doesn't “look inside” the function f . Consequently, the study of optimization algorithms always involves taking advantage of problem structure, whether that structure comes from a particular form for the objective f (as in the clique structure of the MRF objective (1.1)), or from structure in the feasible set \mathcal{X} .

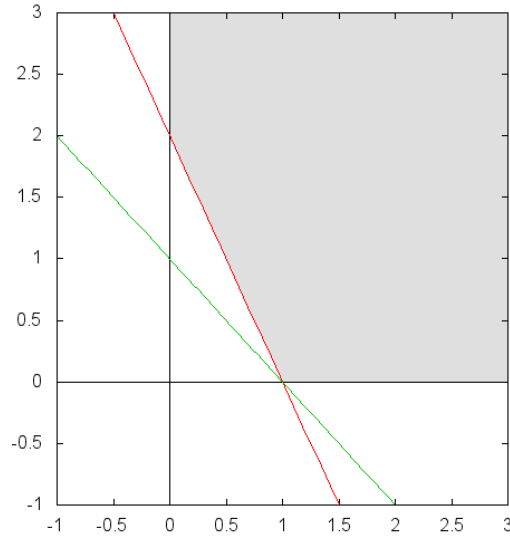


Figure 1.1: The constraints of the optimization problem 1.5 are graphed above. The feasible region is shaded grey. Note that any value which satisfies $2x_1 + x_2 \geq 2$ and $x_2 \geq 0$ also satisfies the inequality $x_1 + x_2 \geq 1$.

1.2.1 Constrained Optimization

Very commonly, problem structure comes from the state space \mathcal{X} being defined by some *constraints*. That is, the set \mathcal{X} is specified by a set of equations or inequalities that the elements $\mathbf{x} \in \mathcal{X}$ must satisfy. For example, in the simple optimization problem

$$\begin{aligned}
 &\min_{x_1, x_2} 3x_1 + 4x_2 \\
 &\text{s.t. } 2x_1 + x_2 \geq 2 \\
 &\quad x_1 + x_2 \geq 1 \\
 &\quad x_1, x_2 \geq 0
 \end{aligned} \tag{1.5}$$

we have 4 constraints, namely that x_1, x_2 satisfy each of the four inequalities: $2x_1 + x_2 \geq 2$, $x_1 + x_2 \geq 1$, $x_1 \geq 0$ and $x_2 \geq 0$. Solutions which violate any of these inequalities are called infeasible. For example $(x_1, x_2) = (\frac{1}{2}, 0)$ is infeasible

(meaning $(\frac{1}{2}, 0) \notin \mathcal{X}$), because in the first inequality we would have $2 \cdot \frac{1}{2} + 0 = 1 < 2$.

In general, in a constrained optimization problem we are given some functions $g_j : \mathcal{X}' \rightarrow \mathbb{R}$ indexed by $j \in J$, and our optimization problem is

$$\begin{aligned} \min_{\mathbf{x}} f(\mathbf{x}) \\ \text{s.t. } g_j(\mathbf{x}) \geq 0 \quad \forall j \in J \\ \mathbf{x} \in \mathcal{X}' \end{aligned} \tag{1.6}$$

We call each inequality $g_j(\mathbf{x}) \geq 0$ a *constraint*, and the *feasible set* \mathcal{X} is defined to be the subset of \mathcal{X}' for which each constraint is *satisfied*, i.e., $\mathcal{X} = \{\mathbf{x} \in \mathcal{X}' \mid g_j(\mathbf{x}) \geq 0, \forall j \in J\}$. The set \mathcal{X}' is called the *ambient space*, and is typically \mathbb{R}^n for some n . Also note that we may have constraints of the form $g_j(\mathbf{x}) \leq 0$ or $g_j(\mathbf{x}) = 0$. These can each be converted into the standard form of (1.6) by noting that $g_j(\mathbf{x}) \leq 0$ is equivalent to $-g_j(\mathbf{x}) \geq 0$, and $g_j(\mathbf{x}) = 0$ can be replaced by the two inequalities $g_j(\mathbf{x}) \geq 0$ and $-g_j(\mathbf{x}) \geq 0$.

Any optimization problem may have many ways of being written in terms of constraints: some constraints may be redundant, for example in (1.5) the constraint $x_1 + x_2 \geq 1$ is redundant, since any x_1, x_2 which satisfies the first inequalities also satisfies $x_1 + x_2 \geq 1$ (see Figure 1.1 for illustration).

1.2.2 Constraint Indicator Functions

An important construction for turning constrained minimization problems into unconstrained minimization is the indicator function: for a minimization prob-

lem and constraint $g_j(\mathbf{x}) \geq 0$, the indicator function is

$$I_{g_j}^{\min}(\mathbf{x}) := \begin{cases} 0 & g_j(\mathbf{x}) \geq 0 \\ \infty & \text{otherwise} \end{cases}. \quad (1.7)$$

That is, $I_{g_j}^{\min}$ is 0 whenever \mathbf{x} satisfies the constraint, and is infinite whenever the constraint is violated. Using the indicator function, we can replace all our constraints with terms in the objective, to get an unconstrained minimization $\min_{\mathbf{x}} F(\mathbf{x})$ where

$$F(\mathbf{x}) = f(\mathbf{x}) + \sum_j I_{g_j}^{\min}(\mathbf{x}) \quad (1.8)$$

Whenever \mathbf{x} is feasible (i.e., satisfies all the inequalities $g_j(\mathbf{x}) \geq 0$) then $\sum_j I_{g_j}^{\min}(\mathbf{x}) = 0$, so $F(\mathbf{x}) = f(\mathbf{x})$. However, if \mathbf{x} is infeasible, it violates at least one inequality, so we will have $F(\mathbf{x}) = \infty$. In other words, we have replaced ‘disallowed’ solutions which violate the constraints, by putting an infinite cost on those solutions. Therefore, minimizing the unconstrained F is the same as minimizing f with the constraints g_j .

Note that for maximization problems, we instead have that solutions with value $-\infty$ are infeasible, so the indicator function for a maximization problem is

$$I_{g_j}^{\max}(\mathbf{x}) := \begin{cases} 0 & g_j(\mathbf{x}) \geq 0 \\ -\infty & \text{otherwise} \end{cases}. \quad (1.9)$$

1.2.3 Minimizing Elements

We will reserve the notation \mathbf{x}^* for elements \mathbf{x} which are optimal, i.e., for which $f(\mathbf{x}^*) = \min_{\mathbf{x}} \{f(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\}$. The set of all such \mathbf{x} is denoted by argmin , so that

$$\operatorname{argmin}_{\mathbf{x} \in \mathcal{X}} \{f(\mathbf{x})\} := \{\mathbf{x} \mid f(\mathbf{x}) \leq f(\mathbf{x}'), \forall \mathbf{x}' \in \mathcal{X}\}. \quad (1.10)$$

We will use the shorthand X^* to denote the set of minimizers, when the problem we are referring to is clear from context. Note that in general the set of minimizers may be empty or have more than one element. For example, $\operatorname{argmin}_{x \in \mathbb{R}} \{e^x\} = \emptyset$ and $\operatorname{argmin}_{x \in \mathbb{R}} \{(x-1)^2(x+1)^2\} = \{-1, 1\}$.

Remark 1. *For almost all problems in this thesis, it will be the case that the set of minimizers is non-empty. We will make special note of problems where this is not necessarily the case. Correspondingly, all proofs will make use of the existence of optimizers where useful.*

There are various conditions which ensure that minimizers exist. One of the most powerful is the following:

Theorem 2. *If X is topologically compact and $f : X \rightarrow \mathbb{R}$ is continuous, then $\operatorname{argmin}_{\mathbf{x} \in X} f(\mathbf{x}) \neq \emptyset$.*

In particular, if X is a finite set, then it is always compact and any function $f : X \rightarrow \mathbb{R}$ is continuous. Therefore, in the common case of $|X|$ finite (also called discrete X) minimizers always exist.

Another special case is when the feasible set X is a subset of \mathbb{R}^n . A subset of \mathbb{R}^n is compact if and only if it is closed and bounded (meaning there is some R with $\|x\| < R$ for all $x \in X$), so in particular minimizers always exist on any closed, bounded subset of \mathbb{R}^n .

1.2.4 Relaxations

An important question is how to relate two optimization problems. The most common relation we will deal with is the notion of *relaxation*. The basic idea

behind relaxations comes from a seemingly trivial observation: if we minimize over a larger set, the minimum value must go down. For example, it's clear that $\min\{3, 8, 6\} \geq \min\{3, 8, 6\} \cup \{1, 5\}$, since the latter, larger set contains the minimum value 3 of the smaller set, plus possibly some other elements which may be smaller (like 1). This idea is the basic intuition for relaxations, in which we take a constrained optimization problem and we ignore, or relax, some constraints. For example, from our example (1.5) above, we can get a relaxation by removing the third constraint (that $x_1 \geq 0$) to get

$$\begin{aligned}
& \min_{x_1, x_2} 3x_1 + 4x_2 \\
& \text{s.t. } 2x_1 + x_2 \geq 2 \\
& \quad x_1 + x_2 \geq 1 \\
& \quad x_2 \geq 0
\end{aligned} \tag{1.11}$$

In this case, we know that $\text{OPT (1.5)} \geq \text{OPT (1.11)}$, since the latter problem is minimizing over a larger set.

To cover all the cases we'll use later on, we'll extend this notion to work not just with constrained optimization. We'll also allow renaming of elements of X , by a function g from X to some other set X' .

Definition 3. *An optimization problem (f, X) embeds into another problem (f', X') if there is a function $g : X \rightarrow X'$ with $f(\mathbf{x}) = f'(g(\mathbf{x}))$ for all $\mathbf{x} \in X$.*

The most important special case of embeddings is relaxation, where $X \subseteq X'$ and the mapping g is just the inclusion map: $g(\mathbf{x}) = \mathbf{x}$. From our example above, we should expect the minimizing value of the relaxed problem to be no larger than the original problem.

First, we have a very basic fact about lower bounds:

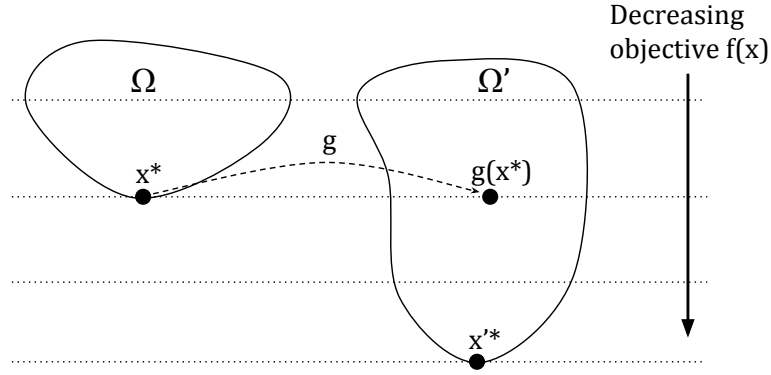


Figure 1.2: Illustration of the proof of Lemma 5. As long as the function g preserves objective values, then whatever the minimizer of (f', \mathcal{X}') is, it is at least as good as $g(x^*)$.

Proposition 4. *If L is a lower bound of a set of real numbers $A \subseteq \mathbb{R}$, meaning $L \leq a$ for all $a \in A$, then $L \leq \min A$.*

Lemma 5. *If (f, \mathcal{X}) embeds into (f', \mathcal{X}') then $\text{OPT}(f, \mathcal{X}) \geq \text{OPT}(f', \mathcal{X}')$.*

Proof. See figure 1.2 for illustration.

We'll show that $\text{OPT}(f', \mathcal{X}')$ is a lower bound to $\{f(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\}$. The Lemma then follows immediately from Proposition 4.

Let $\mathbf{x} \in \mathcal{X}$, we have that $g(\mathbf{x}) \in \mathcal{X}'$ and $f'(g(\mathbf{x})) = f(\mathbf{x})$ since g is an embedding. Then, since $g(\mathbf{x})$ is feasible for (f', \mathcal{X}') we have $f'(g(\mathbf{x})) \geq \text{OPT}(f', \mathcal{X}')$ and therefore $f(\mathbf{x}) \geq \text{OPT}(f', \mathcal{X}')$ as well. \square

1.2.5 Equivalence of Optimization Problems

Another question we might want to ask is: under what conditions can we use one problem to solve another. We can think of such problems as equivalent — a

solution to one gives us a solution to the other, and vice-versa. There are many ways of formalizing this notion of equivalence, but the following will be the most helpful for our purposes.¹

Definition 6. *Two optimization problems (f, \mathcal{X}) and (f', \mathcal{X}') are equivalent if there is a bijection $g : \mathcal{X} \rightarrow \mathcal{X}'$ which is order-preserving, i.e.,*

$$f(\mathbf{x}) \leq f(\mathbf{y}) \quad \text{implies} \quad f'(g(\mathbf{x})) \leq f'(g(\mathbf{y})) \quad (1.12)$$

If g, g^{-1} are poly-time computable, then we'll say (f, \mathcal{X}) and (f', \mathcal{X}') are poly-time equivalent.

Given our identity (1.4) converting maximization problems to minimization problems, that $\min_{\mathbf{x}} f(\mathbf{x}) = -\max_{\mathbf{x}} -f(\mathbf{x})$, we will say that a maximization problem (f, \mathcal{X}) and minimization problem (f', \mathcal{X}') are equivalent if there is an *order-reversing* bijection between them, meaning $f(\mathbf{x}) \leq f(\mathbf{y})$ implies $f'(\mathbf{x}) \geq f'(\mathbf{y})$.

As we claimed, given equivalent problems we can convert solutions of one to solutions of the other. Note that in the case of a poly-time equivalence, this is a reduction in the usual NP-completeness sense.

Lemma 7. *If (f, \mathcal{X}) and (f', \mathcal{X}') are equivalent (with mappings $g : \mathcal{X} \rightarrow \mathcal{X}'$ and $g^{-1} : \mathcal{X}' \rightarrow \mathcal{X}$) the functions g and g^{-1} send minimizers to minimizers: $g(X^*) \subseteq X'^*$ and $g^{-1}(X'^*) \subseteq X^*$.*

Proof. Let \mathbf{x}^* be a minimizer of (f, \mathcal{X}) . We want to show that $g(\mathbf{x}^*)$ is a minimizer of (f', \mathcal{X}') , so let \mathbf{x}' be any other element of \mathcal{X}' . Since g is a bijection, we have that

¹In particular, we define problem equivalence so that we can convert from optimal solutions of one problem to optimal solutions of another problem. When investigating probabilistic inference, this definition is most suited to taking the log-probability of a Gibbs energy to get an energy function of the form (1.1). However, it does not preserve approximation algorithms, and two equivalent problems (according to this definition) may have very different best-possible approximation ratios.

$g^{-1}(\mathbf{x}') \in \mathcal{X}$, and since \mathbf{x}^* is a minimizer of f , we must have

$$f(\mathbf{x}^*) \leq f(g^{-1}(\mathbf{x}')). \quad (1.13)$$

Then, since g is order preserving, we have

$$f'(g(\mathbf{x}^*)) \leq f(g(g^{-1}(\mathbf{x}'))) = f(\mathbf{x}'). \quad (1.14)$$

Therefore, $f'(g(\mathbf{x}^*))$ is less than any other element of \mathcal{X}' , so $g(\mathbf{x}^*)$ is a minimizer of (f', \mathcal{X}') . The reverse claim follows by symmetry. \square

1.2.6 Common Equivalences Between Problems

There are several common transformations we will apply to problems, that all deal with manipulating the objective function to get an equivalent problem. In particular, adding a constant to the objective function or multiplying the objective by a fixed positive constant leads to an equivalent optimization problem. These are both special cases of a general rule: applying a monotonic transformation to the objective is an equivalence.

Lemma 8. *If $g : \mathbb{R} \rightarrow \mathbb{R}$ is monotonically increasing (i.e., $x \leq y$ implies $g(x) \leq g(y)$) then the optimization problems (f, \mathcal{X}) and $(g \circ f, \mathcal{X})$ are equivalent.*

Proof. The identity function $id : \mathcal{X} \rightarrow \mathcal{X}$ is of course a bijection. Furthermore, whenever g is monotonic, then id is order-preserving, in the sense of Definition 6. Indeed, we need to show that

$$f(\mathbf{x}) \leq f(\mathbf{y}) \quad \text{implies} \quad f'(id(\mathbf{x})) \leq f'(id(\mathbf{y})). \quad (1.15)$$

But plugging in $f' = g \circ f$ this is just

$$f(\mathbf{x}) \leq f(\mathbf{y}) \quad \text{implies} \quad g(f(\mathbf{x})) \leq g(f(\mathbf{y})) \quad (1.16)$$

which follows from g being monotonic. □

Corollary 9. *For any constant b and positive constant $a > 0$, the optimization problems (f, X) and $(af + b, X)$ are equivalent.*

Proof. The function $ax + b$ is monotonic for $a > 0$. □

In particular, we can always ignore constant terms in any optimization problem. For example in (1.5) the objective $7 + x_1 + 2x_2$ can be replaced with $x_1 + 2x_2$, which is an equivalent problem according to Lemma 8.

Other useful monotonic functions include \log and \exp which we can combine with Lemma 8 to convert products to sums and vice-versa.

1.3 Example: Image Segmentation

In this section, we will illustrate all of the above concepts by way of an extended example. In computer vision, a prototypical use of optimization is to compute a foreground-background segmentation of an image — most commonly solved by reduction to minimum-cut in a graph. This example is likely familiar to readers familiar with graph cuts; however, it illustrates the key concepts of reduction, gadgets, and the tradeoff between more complex models and efficient optimization, all of which are main themes of this thesis.

The foreground-background segmentation problem is this: for each pixel, we want to give a binary label indicating that this pixel is either part of the foreground object, or part of the background. With n pixels, there are 2^n possible

segmentations, so brute-force searching for the best one is impractical. However, we haven't yet specified f or \mathcal{X} , which together will give us the structure we need to solve the problem.

1.3.1 Binary Labeling Problems

First, we define the feasible set \mathcal{X} . Binary segmentation is an example of a *labeling problem*, where we are trying to give every pixel of an image some discrete label. We formalize this by giving each pixel a corresponding variable. That is, if we let V be the set of pixel indices, then for every $i \in V$ there is a corresponding variable x_i . These variables take values (also called labels) from $\{0, 1\}$, where a label of $x_i = 0$ indicates that pixel i is assigned to the background, and $x_i = 1$ indicates the variable is in the foreground. Therefore, the feasible set (called a label space in a labeling problem) is $\mathcal{X} = \{0, 1\}^V$. In particular, because there are only two labels, this problem is called a *binary* labeling problem.

1.3.2 Per-Pixel Cost Functions

There are many possible choices for the cost function f . We will start with the simplest one first, where we assume that for every pixel, we have some (not necessarily accurate) idea of whether it is likely to be in the foreground or the background. For example, in the image in Figure 1.3, we are trying to segment out the banana as the foreground object, with the remainder of the image as the background. If (for example) a machine learning algorithm has seen many examples of segmented bananas, it could learn that yellow and brown pixels are



Figure 1.3: (Left) An example input to a binary segmentation problem. The foreground object we want to segment out is the banana. (Right) The desired segmentation mask for the foreground object.

likely to be foreground, while other colors are likely to be background. These rough ideas of likelihood are formalized as a cost function f_i for each pixel: $f_i(0)$ gives the cost for the pixel being in the background, and $f_i(1)$ that of the pixel being in the foreground. We then have an objective function

$$f(\mathbf{x}) = \sum_i f_i(x_i). \quad (1.17)$$

If our costs are constructed such that low cost $f(\mathbf{x})$ equates to high likelihood of \mathbf{x} , then choosing the minimizing \mathbf{x} is the most likely solution.

The function (1.17) has a particularly simple structure to optimize: a function is called *separable* when it can be written as a sum of functions $f_i(x_i)$, each of which is a function of a single variable x_i , with no shared variables between them. That is, we can write $f(\mathbf{x}) = \sum_i f_i(x_i)$. In this case, it is clear that we can find the minimizing \mathbf{x} by setting x_i to be 0 if $f_i(0) \leq f_i(1)$, and 1 otherwise. For separable objectives, making locally good choices gives a globally optimal solution, so these are among the simplest objectives to optimize.



Figure 1.4: (Left) The resulting segmentation using a unary-only model, as in (1.17). (Right) The resulting segmentation after adding edge terms of the form (1.19). Note that the segmentation boundaries much more closely follow the actual boundaries of the foreground object.

1.3.3 Spatial Relations Between Pixels

This simple, separable model unfortunately does not give good results — even with very informative functions f_i , the segmentations obtained tend to be very noisy, as shown in Figure 1.4. The problem is that our model is missing important knowledge about the problem: it makes the (implicit) assumption that the label of a pixel is unconnected with the label of its neighbors. We can see by noting that the minimizing x_i is found independently of the others (i.e., without reference to f_j for $j \neq i$). However, we actually know a lot about the relations between pixels in an image. For example, the foreground labels tend to form a connected region in the image, and in general, a given pixel being in the foreground is good evidence that its neighbors are likely to be foreground pixels as well (and similarly for background pixels).

The intuition we want to incorporate into improving (1.17) is that we should take advantage of the spatial relations between pixels in an image. To do so, we

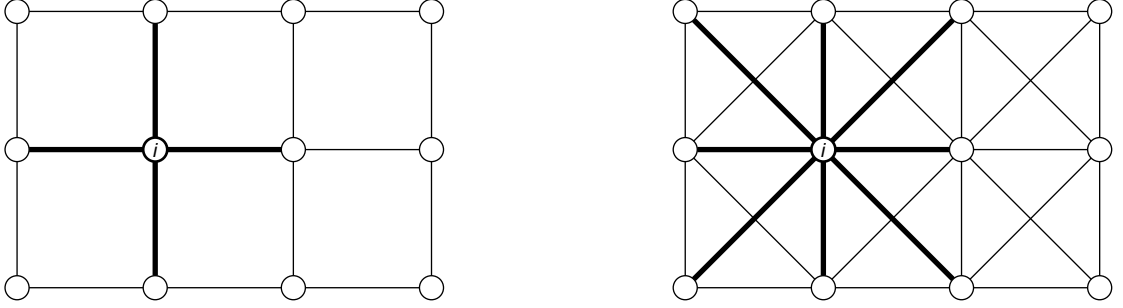


Figure 1.5: (left) Typical 4-connected neighborhood on a pixel grid (right) 8-connected neighborhood. Neighbors $N(i)$ of pixel i are bolded.

make use of the fact that the pixels V are arranged in a grid, and we will say that pixels i, j are neighbors if they are nearby in this grid. Typical neighbor sets are the 4-connected grid (where i has up to 4 neighbors up, down, left and right of it) and the 8-connected grid (which includes the 4-connected neighbors, as well as the 4 immediately diagonal neighbors). See Figure 1.5 for illustration. We will let $N(i)$ be the set of pixels j which are neighbors of i , and let $E = \{\{i, j\} \mid j \in N(i)\}$ be the set of all pairs of neighbors, called edges.

1.3.4 The Potts Model

With this neighborhood structure, a simple model which encourages neighbors to take the same labels is the Potts model [75]. This model has the *unary costs* f_i from (1.17), as well as *pairwise costs* $f_{i,j}$ between pairs of neighbors $\{i, j\} \in E$. The pairwise costs are of a particularly simple form — we pay a flat cost every time the endpoints of an edge i, j have different labels. That is,

$$f_{i,j}(x_i, x_j) = \begin{cases} 1 & x_i \neq x_j \\ 0 & \text{otherwise} \end{cases}. \quad (1.18)$$

Note that we'll index $f_{i,j}$ using unordered pairs, meaning $f_{i,j}(x_i, x_j) = f_{j,i}(x_j, x_i)$.

We incorporate these pairwise terms into (1.17) by letting

$$f(\mathbf{x}) = \sum_i f_i(x_i) + \lambda \sum_{i,j} f_{i,j}(x_i, x_j). \quad (1.19)$$

Because we now pay a cost for neighbors with different labels, the minimum cost solution will tradeoff choosing likely foreground-background assignments, while also not having too many discontinuities in the labeling. We can adjust the parameter λ to adjust the relative weights in this tradeoff. Higher λ encourages more cohesiveness in the segmentation: as $\lambda \rightarrow \infty$ the segmentation will eventually become a single all-foreground or all-background region; as $\lambda \rightarrow 0$ the objective becomes the same as the unary-only model of (1.17), and we get the noisy segmentations seen in Figure 1.4. The best segmentations will be obtained for λ somewhere in the middle, and the best λ must be tuned as a *hyperparameter* of the model (typically by evaluating many such λ on a validation set, and choosing the value with the best segmentation results on this set of held-out images).

1.3.5 Reduction to Graph Cut

However, unlike (1.17), it is not immediately obvious how to find a minimum energy solution \mathbf{x} to (1.19). We cannot simply set each variable x_i to the one with smallest $f_i(x_i)$ as before, as this might cause a large penalty from pairwise terms $f_{i,j}(x_i, x_j)$.

To solve this minimization problem we will use the standard optimization technique of reduction, whereby we transform (or reduce) some seemingly difficult problem into another problem which we already know how to solve. Minimizing (1.19) can be reduced to the well-known optimization problem known

as minimum cut.

The min-cut problem is the following: we are given a graph with nodes N and arcs ² $A \subseteq N \times N$, along with *capacities* $c_{i,j}$ on each arc $(i, j) \in A$. There are two special nodes $s, t \in V$ called respectively the source and sink. A cut is a set $S \subseteq N$ that contains s but not t (i.e., $s \in S$ and $t \notin S$). The cost of a cut is the sum of all capacities of arcs with their first endpoint in S , and the other outside S , so $c(S) = \sum_{i \in S} \sum_{j \notin S} c_{i,j}$. Our goal is to find a cut S minimizing $c(S)$.

There exist many efficient algorithms for solving these problems, including Augmenting Paths [23], Push Relabel [14], as well as the current state-of-the-art for inputs typical of vision problems: IBFS [30]. If we can transform our actual problem (1.19) into an input for the min-cut problem, then we can apply any of these algorithms to find the solution. Fortunately, the min-cut problem has many similarities to the binary segmentation problem. If we think of membership-or-not in S as a binary label, then the cost function $c(S)$ looks very much like the Potts terms $f_{i,j}$: we pay a cost $c_{i,j}$ whenever neighbors i and j take different labels (i.e., i is in S and j is not in S). The only difficulty is handling the unary terms f_i .

To transform our problem into a min-cut input, we introduce a *graph construction* (sometimes called a gadget), which is a method for constructing a particular graph whose vertices and edges are chosen in such a way that finding the minimum cut in this graph gives a solution to the binary segmentation problem. For Binary Segmentation, our constructed graph has a vertex for every pixel of the original problem, as well as two special nodes s and t , called a super-source and super-sink, so $N = V \cup \{s, t\}$.

²We distinguish *edges*, which are undirected (i.e., are unordered pairs $\{i, j\}$), from *arcs* which are directed pairs (i, j) .

For every undirected edge $e = \{i, j\}$ in the neighborhood structure, we get two directed arcs (i, j) and (j, i) , both of which have capacity λ . Finally, we account for the unary terms by adding arcs (s, i) and (i, t) for every pixel i , with capacities $c_{s,i} = f_i(0)$ and $c_{i,t} = f_i(1)$.

This is now a valid input to the min-cut problem, so the only remaining question is whether this is actually an *equivalent* optimization problem to our original objective (1.19). According to Definition 6, we need an order-preserving bijection between these two problems. We've already noted a natural way of identifying binary vectors \mathbf{x} with cuts S , namely, taking membership in S as a binary label. To ensure that s is always in the cut S and t is not in S , we define g from binary labelings to cuts by $g(\mathbf{x}) = \{s\} \cup \{i \in V \mid x_i = 1\}$. It is clear that this is a bijection, with inverse $g^{-1}(S)$ being the binary vector with $x_i = 1$ for $i \in S$ and $x_i = 0$ for $i \notin S$.

Finally, g is not just order preserving, but actually leaves the objective value the same. Indeed, for any \mathbf{x} we have

$$f(\mathbf{x}) = \sum_{i: x_i=1} f_i(1) + \sum_{i: x_i=0} f_i(0) + \sum_{i,j: x_i=1, x_j=0} \lambda \quad (1.20)$$

whereas the cost of the corresponding cut S is

$$c(S) = \sum_{i \in S} f_i(1) + \sum_{i \notin S} f_i(0) + \sum_{i,j: i \in S, j \notin S} \lambda \quad (1.21)$$

Then, remembering that $x_i = 1$ if and only if $i \in S$, we see that these two equations are the same, so $f(\mathbf{x}) = c(S)$.

1.3.6 Discussion

It's important to not let these details obscure the overall plan at work — because the binary segmentation problem is equivalent to the min-cut problem, Lemma 7 says we can take any optimal solution to the min-cut problem and get back an optimal solution to the segmentation problem. In fact, this solution is as simple as taking the min-cut S^* , and then applying g^{-1} to get a binary map x^* out of it.

A final observation: the min-cut problem was already close enough to our segmentation problem that we should actually construct a gadget showing them to be equivalent. This is no coincidence — only by choosing to model the binary cut problem in this particular way (with an objective function of the form (1.19)) is this connection obvious.

In fact, the existence of this graph construction is somewhat fragile — as we add more detail to the model, with more elaborate cost functions, it may break. For example, if we want to modify (1.19) to allow having a different cost $\lambda_{i,j}$ on each edge, then we are still fine: the constructed min-cut problem would now have a different cost $c_{i,j} = \lambda_{i,j}$ on each internal arc (i, j) , and min-cut will still find a solution. However, if we want to solve more general segmentation problems, for example *semantic segmentation*, where the label space is a larger set of semantic categories (e.g., sky, grass, building, etc.) then it is not obvious how to repair the construction, since min-cut is a binary problem, and multi-label generalizations of min-cut (such as multi-way cut) are generally NP-hard. Another change we could consider making is to allow some pixels to have a *negative* cost for taking different labels (i.e., allowing $\lambda_{i,j} < 0$). In this case, our construction would have negative capacities $c_{i,j} < 0$, and the minimization problem again

NP-complete (by reduction from max-cut).

This difficulty is the heart of the tension between modeling and inference — the limits of our inference algorithms constrain the expressiveness of how we can model problems from applications.

1.4 Markov Random Fields

Having seen how these ideas work in a practical example, we can now formalize all of the concepts that make foreground-background segmentation work. The following definitions are all building towards the main topic of this thesis: optimization for Markov Random Fields.

1.4.1 Labeling Problems

In our segmentation example, our optimization problem had a variable x_i for each pixel i , and each variable could take one of two labels, representing a choice of foreground or background for that pixel. Many problems have this general form, frequently with larger label sets than just $\{0, 1\}$. For example, in a semantic segmentation problem, the pixels take labels from a pre-defined set of semantic categories, such as Sky, Ground, Tree, etc. The label sets can be quite large in practice: in optical flow each pixel is labeled with a two-dimensional displacement, labeling every pixel with its corresponding pixel in the previous video frame, so that the 2 pixels belong the same physical point — in this case the size of the label space is potentially the total number of pixels in the image (and even larger if subpixel displacements are desired).

Definition 10. A labeling problem is an optimization problem with a set of variables x_i indexed by $i \in V$ each taking values from a label set \mathcal{X}_i . That is, $\mathcal{X} = \prod_{i \in V} \mathcal{X}_i$.

We will use vector notation $\mathbf{x} = (x_1, \dots, x_n)$ to indicate a feasible state, which we will call a labeling. We will write \mathbf{x}_C for the subvector of \mathbf{x} restricted to the indices $i \in C$, so that $\mathbf{x}_C = (x_i)_{i \in C}$. Similarly, define $\mathcal{X}_C = \prod_{i \in C} \mathcal{X}_i$ to be the subspace of \mathcal{X} corresponding to C .

1.4.2 Maximum A-Posteriori (MAP) Inference

In defining the unary terms for equation (1.17), we indicated that the f_i were chosen to correspond to how *likely* a particular pixel was to be background or foreground, given the information we observed in the image (i.e., that a particular yellow pixel was likely to be part of the banana, and therefore foreground). This rough idea is formalized by the notion of *probabilistic inference*.

For probabilistic inference, in addition to our unknown state $\mathbf{x} \in \mathcal{X}$, we also have a set of observations $\mathbf{y} \in \mathcal{Y}$. For example, in binary segmentation our observations include all the color values y_i of each pixel i that form the image.

Probabilistic inference further assumes that there is some joint probability distribution $p \in \mathcal{P}[\mathcal{X} \times \mathcal{Y}]$ over all possible observations and their associated labelings. We don't assume we have knowledge of this joint probability distribution, but it does allow us to consider the conditional probability $p[\mathbf{x}|\mathbf{y}]$: the probability of an unknown state $\mathbf{x} \in \mathcal{X}$, given that we observed $\mathbf{y} \in \mathcal{Y}$. This conditional distribution is known as the posterior probability (since it is the remaining probability *after* having observed a definite \mathbf{y}). The Maximum A-Posteriori

(MAP) problem is to find the state \mathbf{x} with highest posterior probability

$$\max_{\mathbf{x} \in \mathcal{X}} p[\mathbf{x}|\mathbf{y}] \quad (1.22)$$

That is, having seen (i.e., conditioning on) the values of the observable \mathbf{y} , we choose the most probable \mathbf{x} as our answer. Note that MAP is not the only probabilistic decision method, but it is empirically quite successful, as well as amenable to an optimization approach.³

In order to compute this posterior probability, it is frequently easier to reason about another conditional distribution: $p[\mathbf{y}|\mathbf{x}]$. This is known as the likelihood function, and gives the probability of seeing an observation given that the true state was \mathbf{x} . Likelihood functions are useful whenever we have a forward model of how our observations are formed from the hidden latent variables \mathbf{x} .

Given this likelihood function, we can use Bayes' rule to compute the posterior distribution

$$p[\mathbf{x}|\mathbf{y}] = \frac{p[\mathbf{y}|\mathbf{x}]p[\mathbf{x}]}{p[\mathbf{y}]} \quad (1.23)$$

There are two other terms to explain. The probability $p[\mathbf{x}]$ is the unconditional probability that the true state is \mathbf{x} . Since this probability does not depend at all on our observation, it is known as the *prior* probability of the state \mathbf{x} . The probability $p[\mathbf{y}]$ is the unconditional probability of a particular observation \mathbf{y} , regardless of the true state \mathbf{x} . Since in MAP inference we are optimizing over \mathbf{x} , and \mathbf{y} is fixed, this is a fixed positive constant multiplied to our objective, so given Corollary 9 (on problem equivalence), we are free to ignore it. In contexts

³It's worth noting that MAP inference is a Bayes Optimal decision procedure, only when the loss function is 0-1, meaning we only care if we have found the exact correct solution and not any nearby point. In practice, we often have a loss function which allows some amount of inaccuracy; however, the decision procedure in this case often requires more difficult (and frequently intractable) inference algorithms, such as finding the posterior marginal distributions for each variable.

where \mathbf{y} can vary, this probability $p[\mathbf{y}]$ is usually denoted Z and is called the partition function.

An important point is that for almost all applications, the true underlying distribution $p[\mathbf{x}, \mathbf{y}]$ by which observations and hidden states are generated is both unknown and likely unknowable, and in any case much too complicated an object to perform any computation with. However, we may have a reasonable model, which approximates this probability distribution, and which is close enough to the true distribution to give reasonable answers.

1.4.3 Log-probabilities

A useful transformation for probabilistic inference is to consider the negative log of the probability, converting the product in (1.23) into a sum

$$-\log(p[\mathbf{x}|\mathbf{y}]) = -\log(p[\mathbf{y}|\mathbf{x}]) - \log(p[\mathbf{x}]) + \log(Z). \quad (1.24)$$

Because $-\log$ is monotonically decreasing on $[0, \infty)$, it is an order-reversing bijection, so the minimization problem

$$\min_{\mathbf{x}} -\log(p[\mathbf{x}|\mathbf{y}]) = \min_{\mathbf{x}} [-\log(p[\mathbf{y}|\mathbf{x}]) - \log(p[\mathbf{x}]) + \log(Z)] \quad (1.25)$$

is equivalent to our original probabilistic inference problem (1.22), by Lemma 8. This form has a number of advantages over (1.22). Because optimization problems are invariant up to addition of constants, we can ignore the term $\log Z$ entirely. Then, define

$$f_{\text{data}}(\mathbf{x}) := -\log(p[\mathbf{x}|\mathbf{y}]) \quad (1.26)$$

$$f_{\text{prior}}(\mathbf{x}) := -\log(p[\mathbf{x}]). \quad (1.27)$$

Then the MAP problem is equivalent to

$$\min_{\mathbf{x}} f_{\text{data}}(\mathbf{x}) + f_{\text{prior}}(\mathbf{x}) \quad (1.28)$$

The negative log transformation therefore lets us separate the minimization into two parts, one dealing with how well a state \mathbf{x} fits the observed data \mathbf{y} , and another dealing only with the prior probability of \mathbf{x} .

If some of our probabilities are independent of each other, we can simplify this even further. Many models assume that each observation y_i is generated independently from the other y_j , and in fact only depends on a single unknown x_i . In this case, we have

$$p[\mathbf{y}|\mathbf{x}] = \prod_i p[y_i|x_i]. \quad (1.29)$$

We will say that such a model has *separable likelihood*, because we can take the negative log of these probabilities, and define $f_i(x_i) := -\log(p[y_i|x_i])$, to get

$$f_{\text{data}}(\mathbf{x}) = \sum_i f_i(x_i). \quad (1.30)$$

This is a separable function of the variables x_i . In this case, the functions f_i are called unary data terms, since they are functions of a single variable, depending only on a single piece of data.

1.4.4 MAP inference in Foreground-Background Segmentation

Returning to our example of Foreground-Background Segmentation, we can now be explicit about the unary terms f_i in (1.17), and how they relate to likelihood of pixels taking certain labels. As we have seen, we can use Bayes rule to calculate the posterior probability, but we need a likelihood function $p[\mathbf{y}|\mathbf{x}]$ and prior $p[\mathbf{x}]$.

In reality, there is no simple rule which takes a foreground-background labeling \mathbf{x} and gives a probability of different images \mathbf{y} with that particular segmentation. The space of “all possible images” is much too large to specify a distribution over, and moreover requires a distribution over images collected from the real world.

We can make a reasonable approximation by choosing a much simpler, but still plausible choice for our likelihood function. In foreground-background segmentation, one common choice is to use a *color model*, where we assume that the color of objects is drawn from a population, and we have distinct populations for the foreground and background objects.

Specifically, we will choose for our unary likelihood terms a Gaussian Mixture Model (GMM), which models the individual pixels y_i as independently chosen, and picked according to weighted sums of gaussians. The gaussians are different depending on whether the hidden state x_i is foreground ($x_i = 1$) or background ($x_i = 0$).

Definition 11. *A Gaussian Mixture Model is a distribution over \mathbb{R}^n which is a weighted sum of gaussians. A GMM has probability distribution function*

$$GMM(c) = \sum_{i=1}^k w_i N_{\mu_i, \Sigma_i}(c) \quad (1.31)$$

for a color $c \in \mathbb{R}^3$, where w_1, \dots, w_k are weights summing to 1, and N_{μ_i, Σ_i} are normal distributions with mean μ_i and covariance Σ_i .

Note that RGB colors come from \mathbb{R}^3 , so we are using 3-dimensional gaussians. The GMM model for segmentation has 2 different distributions, GMM_{FG} and GMM_{BG} . Each pixel y_i in the image (part of our observations) is generated independently of the other pixels: if the corresponding label x_i is foreground,

then it is drawn from the distribution GMM_{FG} , if x_i is background, it is instead drawn from GMM_{BG} . Therefore, the likelihood function is

$$p[y_i|x_i] = \begin{cases} \text{GMM}_{\text{FG}}(y_i) & x_i = 1 \\ \text{GMM}_{\text{BG}}(y_i) & x_i = 0 \end{cases} \quad (1.32)$$

Taking negative log probabilities, we can define unary terms

$$f_i(\mathbf{x}) := -\log(p[y_i|x_i]) \quad (1.33)$$

$$= \begin{cases} -\log(\text{GMM}_{\text{FG}}(y_i)) & x_i = 1 \\ -\log(\text{GMM}_{\text{BG}}(y_i)) & x_i = 0 \end{cases} \quad (1.34)$$

$$= -\log(\text{GMM}_{\text{FG}}(y_i))\delta_1(x_i) - \log(\text{GMM}_{\text{BG}}(y_i))\delta_0(x_i) \quad (1.35)$$

where $\delta_z(x_i)$ is the delta function, defined to be 1 whenever $x_i = z$ and 0 otherwise. Then, the probabilistic inference problem with this choice for the likelihood function becomes

$$\min_{\mathbf{x}} \sum_i f_i(x_i) - \log(p[\mathbf{x}]) \quad (1.36)$$

GMMs are an effective choice for the likelihood function since they capture the intuition of images being composed of a few like-colored objects. For example, in a scene composed of a white cow on a green field with a blue sky, we will have 2 clusters of colors for background pixels, centered around blue and green respectively, and a single cluster of foreground pixels centered around white.

1.4.5 Conditional Dependence

One feature we have not discussed is the choice of prior $p[\mathbf{x}]$. The first thing to note is that if $p[\mathbf{x}]$ is the uniform prior over all images, then $\log(p[\mathbf{x}])$ is a

constant, so can be removed from (1.36). Doing so, we get back the unary-only model of (1.17). Therefore, the separable optimization problem we discussed earlier is exactly the case where we assume no prior over possible segmentations \mathbf{x} — every possible labeling is assumed to be equally likely.

Of course, this is not a very realistic prior — as we have discussed, pixels share a lot of information with their neighbors. Using this information, in the form of a prior $p[\mathbf{x}]$, leads us to our next topic, which is conditional dependence between variables.

In computer vision, conditional dependence between variables is most closely related to spatial locality in images. That is, information at a particular pixel i is strongly correlated with that of its neighbors $j \in N(i)$. In particular, we are interested in probability distributions for which variables x_i depend only on their neighbors x_j for $j \in N(i)$. Such distributions are called Markov [35].

The Markov property relates two different conditional distributions. The first is the probability of a given variable x_i taking the label $a_i \in \mathcal{X}_i$, conditioned on all other variables \mathbf{x}_{V-i} having some already determined labeling \mathbf{a}_{V-i} . The second conditional probability is that of x_i taking a_i , conditioned on just the variables neighboring i , $\mathbf{x}_{N(i)}$, having labels $\mathbf{a}_{N(i)}$.

Definition 12. A probability distribution $p \in \mathcal{P}[\mathcal{X}]$ is Markov (with respect to the neighbor structure N) if (1) we have $p[\mathbf{x}] > 0$ for all $\mathbf{x} \in \mathcal{X}$ and (2) for each i , and every labeling $\mathbf{a}_V \in \mathcal{X}$, we have

$$p[x_i = a_i | \mathbf{x}_{V-i} = \mathbf{a}_{V-i}] = p[x_i = a_i | \mathbf{x}_{N(i)} = \mathbf{a}_{N(i)}]. \quad (1.37)$$

Such a distribution is called a Markov Random Field.

That is, if we specify the labels $x_j = a_j$ for all the neighbors $j \in N(i)$, then the

probability that $x_i = a_i$ is the same, regardless of the labels of any non-neighbor. The variables x_i and x_k for $k \notin N(i)$ are *conditionally independent*, since after conditioning on the neighbors x_j for $j \in N(i)$, the variables x_i and x_k are independent. The Markov property is a particularly strong form of image locality, in that it says non-neighbors have no direct effect on a variable x_i .

1.4.6 The Hammersley-Clifford Theorem

While we are interested in the Markov property because it captures our intuition of spatial locality, we get lucky — all such distributions can be written in a particularly simple form. In particular, the probability is a product over subsets of the variables called *cliques*.

Definition 13. For a neighborhood graph N , the set of (Markov) cliques C consists of all fully-connected subgraphs

$$C := \{C \subseteq V \mid \{i, j\} \in E, \forall i, j \in C\}. \quad (1.38)$$

When we have a function $f_C(\mathbf{x}_C)$ that depends only on \mathbf{x}_C , (i.e., $f_C : \mathcal{X}_C \rightarrow \mathbb{R}$) we will call f_C a clique function.

Theorem 14 (Hammersley-Clifford [35]). A probability distribution p is a Markov Random Field if and only if it can be written in the form

$$p[\mathbf{x}] = \prod_{C \in \mathcal{C}} e^{-f_C(\mathbf{x}_C)} \quad (1.39)$$

where each f_C is a clique function depending only on \mathbf{x}_C . Such a distribution is called a Gibbs distribution.

Therefore, given a posterior distribution $p[\mathbf{x}|\mathbf{y}]$ which is Markov, we can take negative log-probabilities to get a simple form for the MAP problem:

$$\begin{aligned} -\max_{\mathbf{x}} \log p[\mathbf{x}] &= \min_{\mathbf{x}} -\log \left(\prod_C e^{-f_C(\mathbf{x}_C)} \right) \\ &= \min_{\mathbf{x}} \sum_{C \in \mathcal{C}} f_C(\mathbf{x}_C) \end{aligned} \tag{1.40}$$

Since the MAP problem for any Markov Random Field can be written in the form (1.40) (and vice versa), we will henceforth take this as our basic definition of an MRF.

Definition 15. Let \mathcal{C} be a subset of 2^V , i.e., any collection of subsets of V , $\mathcal{X} = \prod_i \mathcal{X}_i$ a label space, and $f : \mathcal{X} \rightarrow \mathbb{R}$ be of the form

$$f(\mathbf{x}) = \sum_{C \in \mathcal{C}} f_C(\mathbf{x}_C). \tag{1.41}$$

Then f is called the MAP problem for the MRF f , or an MRF inference problem with clique structure \mathcal{C} and clique functions f_C .

Note that we don't require the cliques \mathcal{C} to line up with any particular set of fully connected subsets, or to necessarily come from the probability of any probabilistic inference problem. Because the inference methods described in later chapters apply to any such functions, we will refer to any function of the form (1.41) as an MRF.

Because many of the problems we deal with have a separable likelihood, we will generally separate out the unary terms in (1.41), and write f as

$$f(\mathbf{x}) = \sum_{i \in V} f_i(x_i) + \sum_{C \in \mathcal{C}} f_C(\mathbf{x}_C). \tag{1.42}$$

Remark 16. We will generally assume that the cliques are small, relative to the number of variables: $|\mathcal{C}| \ll n$.

In this case, the cliques represent *local structure*, in that they are small collections of non-independent variables.

1.4.7 The Potts Model as an MRF

Returning to our example of foreground-background segmentation, we can now fit the Potts model of Section 1.3.4 into this framework. Recall equation (1.19) for the binary segmentation energy

$$f(\mathbf{x}) = \sum_i f_i(x_i) + \sum_{i,j} f_{i,j}(x_i, x_j). \quad (1.43)$$

As we've already seen, the unary terms f_i come from our probabilistic inference framework via the likelihood functions $p[y_i|x_i]$. The pairwise terms, $f_{i,j}$, however, do not involve the observations \mathbf{y} , and are therefore part of the prior, $p[\mathbf{x}]$.

Note that (1.43) is a particular instance of (1.41), i.e., (1.43) defines an MRF. Here, the clique structure includes the neighbor pairs $\{i, j\} \in E$ for the pairwise terms and the singletons $\{i\}$ for the unary terms:

$$C = \{\{i, j\} | i \in N(j)\} \cup \{\{i\} | i \in V\} \quad (1.44)$$

We can even give this particular choice of pairwise functions $f_{i,j}$ a probabilistic interpretation, by reversing the negative-log transformation we used to get (1.40).

$$p[\mathbf{x}|\mathbf{y}] \propto \prod_i e^{-f_i(x_i)} \prod_{i,j \in E} e^{-f_{i,j}(x_i, x_j)}. \quad (1.45)$$

Note that we only get the probability up to proportionality — since the minimization problem is the same up to an additive constant, reversing the negative-log transformation we only get the probability up to a positive multiplicative constant.

Recalling that $f_{i,j}(x_i, x_j) = \lambda$ if $x_i \neq x_j$ and 0 otherwise, we can simplify the part of 1.45 dealing with the prior $p[\mathbf{x}]$ to

$$p[\mathbf{x}] \propto \prod_{i,j \in E} e^{-f_{i,j}(x_i, x_j)} = \prod_{i,j \in E: x_i \neq x_j} e^{-\lambda}. \quad (1.46)$$

This distribution $p[\mathbf{x}]$ is also known as the Ising model, which has been independently studied in physical models of spin states. In general, such Gibbs distributions [28] (distributions which are products of exponentials) are widely studied in statistical physics and other fields.

1.5 First-order and Higher-order MRFs

One of the most important factors for the complexity of solving an MRF inference problem is the *order* of the MRF.

Definition 17. *The order of an MRF is the maximum size of any clique $C \in \mathcal{C}$, minus one. We distinguish two cases: first-order MRFs, which have maximum clique size two, and higher-order MRFs, which have cliques of size three or greater.*

First-order MRFs were historically the first to have efficient inference algorithms. According to the definition of a first-order MRF, every clique has size at most 2, so is either a unary term, $|C| = 1$, or involves a pair of variables, $|C| = 2$. Therefore, for first order MRFs, we can form a graph $G = (V, E)$ with (unordered) edges $e = \{i, j\}$, such that

$$f(\mathbf{x}) = \sum_i f_i(x_i) + \sum_{\{i,j\} \in E} f_{i,j}(x_i, x_j) \quad (1.47)$$

Because the clique structure \mathcal{C} forms a graph, we can employ existing graph algorithms for this problem. As seen in our example of binary segmentation, (1.47) is closely related to the min-cut problem in a graph, and can be

exactly solved for binary problem. For multi-label problems, we can approximately solve the problem by repeated application of min-cut using algorithms such as alpha-expansion, which we will describe later.

In contrast to first-order MRFs, it is harder to develop efficient optimization algorithms for higher-order MRFs. The first difficulty is that even specifying the values of a clique function $f_C(\mathbf{x}_C)$ for each of the labelings $\mathbf{x}_C \in \mathcal{X}_C$ requires storing $|\mathcal{X}_C|$ values (where $\ell = |\mathcal{X}_i|$). Additionally, whereas a first-order MRF naturally forms a graph, a higher-order MRF forms a hypergraph, where instead of edges consisting of pairs of vertices, we have *hyperedges* which are subsets of the vertices, of size possibly larger than 2. While many graph algorithms can be extended to work on hypergraphs, it is not always completely obvious how to do so. In particular, a generalized version of the min-cut algorithm for hypergraphs was only very recently applied to higher-order MRF optimization [53].

1.5.1 Advantages of Higher-Order Models

Despite these difficulties, higher-order MRFs have a number of advantages, the most important of which is that first-order MRFs are limited in expressiveness compared to general MRFs.

The primary advantage of using higher-order MRFs is that they allow greater flexibility in coming up with models that better match the true statistics of images. Probabilistic inference frequently makes simplifying assumptions about distributions in order to end up at a tractable optimization problem. If we are ultimately interested in more accurate answers, then we require more complicated models that better represent the true prior and likelihood functions.

Compare the different segmentation results from the unary-only model of (1.17) versus the pairwise Potts model of (1.19), as seen in Figure 1.4. In our probabilistic inference framework, the unary-only model corresponds to a MAP problem with a totally uniform prior, $p[\mathbf{x}] = \frac{1}{|\mathcal{X}|}$. This prior does not represent real-world segmentations well, and consequently the results we see in Figure 1.4 are noisy, with boundaries that do not closely match the actual object. By adding in the pairwise Potts terms, we have complicated the model (requiring a reduction to min-cut to solve, rather than the simple separable optimization of (1.17)); however, the observed segmentations correspond much more closely to what we expect real objects to look like. This was achieved by adding a prior $p[\mathbf{x}]$ which explicitly prefers segmentations with a short boundary (a quality which is shared with the true distribution of objects in images).

Compared to first-order models, higher-order MRFs allow even more flexibility to match the underlying distribution, and in many cases can express properties of images that are inexpressible in first-order models. We will consider two examples, patch-based priors for denoising and curvature regularizing priors for stereo.

1.5.2 Image Denoising and Patch-Based Priors

A common problem in photography is that images suffer from noise — low light images in particular suffer from photon noise due to the quantized nature of light, thermal noise, and read-noise from imperfect electronics.

In the image denoising problem, we attempt to remove this noise from an image, in order to reconstruct the underlying scene being imaged. We have a

forward model of image formation, where we observe a noisy image \mathbf{y} , which is obtained from an underlying (noise-free) image \mathbf{x} by the addition of a noise function η to each pixel:

$$y_i = x_i + \eta. \quad (1.48)$$

In our forward model, we will assume that η is independent for each pixel i , and that η is unbiased Gaussian noise, $\eta \sim N(0, \sigma)$. For concreteness, we'll assume that the label space \mathcal{X}_i is discretized to 256 intensity values $\mathcal{X}_i = \{0, \dots, 255\}$.

Our likelihood function is determined by our forward model. Since $y_i = x_i + N(0, \sigma)$, we have

$$p[y_i|x_i] = e^{\frac{-(y_i-x_i)^2}{2\sigma^2}}. \quad (1.49)$$

Taking negative-log probabilities (as in Section 1.4.3) we get unary terms

$$f_i(x_i) = -\log p[y_i|x_i] = \frac{1}{2\sigma^2}(x_i - y_i)^2. \quad (1.50)$$

Note that this is an example of a separable likelihood function.

Without any prior on our problem, the optimization problem is just

$$\min_{\mathbf{x}} \frac{1}{2\sigma^2}(\mathbf{x} - \mathbf{y})^2 \quad (1.51)$$

so the minimizing answer is just to set $\mathbf{x} = \mathbf{y}$ — that is, if we have unbiased noise, and no prior beliefs about noise-free images, the MAP answer is to say that the original noise-free image is whatever we observed.

Of course, noisy images are quite different from noise-free images. In particular, we expect images to be composed of connected regions formed by individual objects (which may be actual objects, or patches of similar color within an object such as the spots on a cow) where the variation of intensity within each object is roughly constant, and with a few sharp transitions where objects meet.

We can capture this intuition with an *edge-preserving prior*, which is a first-order MRF model. For our clique structure C , we'll use either the 4- or 8-connected neighborhood model of Figure 1.5, and let the clique structure C be all edges $C = E$. On each edge $\{i, j\}$ we will have what is called a robust distance function $f_{i,j}$ of the form

$$f_{i,j}(x_i, x_j) = \min\{|x_i - x_j|, \tau\}. \quad (1.52)$$

This specific cost is called the truncated L_1 cost. To explain this choice, note that it has minimum cost when $x_i = x_j$, with gradually increasing cost as x_i and x_j have different intensities. This represents the part of our intuitive description that within an object, neighboring pixels should have similar intensities. Then, because we expect there to be edges between objects where the intensity can jump arbitrarily, we cap the maximum cost at τ .

These edge-preserving priors do a reasonable job matching the observed statistics of neighboring pixel differences, and form a first-order MRF, so we have fast algorithms for optimizing them. However, they fail to take into account more complicated statistics of images, especially those related to image texture. In particular, the truncated L_1 cost always prefers neighbors to have identical intensity values, so totally flat regions will always be preferred to regions of small texture variations.

It is difficult to come up with a texture-aware first-order MRF. However, with higher-order MRFs it is relatively easy. Instead of putting a cost $f_{i,j}$ on pairs of neighboring pixels, we will put a cost f_C on *patches* of the image (such priors are called *patch-based priors*). These patch functions f_C are designed to match the statistics of noise-free image patches, and can thus account for local intensity variation due to texture (among other features). We will focus on a particular



Figure 1.6: From left to right: (a) Original image, before noise. (b) Image after adding independent Gaussian noise to each pixel. (c) Denoised result using the pairwise edge-preserving prior of (1.52). (d) Denoised result using the Field-of-Expert prior of (1.53).

prior known as Field of Experts [77], which consists of a set of k linear patch filters J_i run through a non-linear response function to give a cost function

$$f_C(\mathbf{x}_C) = \sum_{i=1}^k \alpha_i \log(1 + J_i^T \mathbf{x}_C). \quad (1.53)$$

This cost function is not as clearly intuitive as the edge-preserving truncated L_1 cost; however, we can use machine learning techniques so that the filters J_i and weights α_i are chosen such that the resulting prior $p[\mathbf{x}]$ matches the observed distribution of image patches as closely as possible.

The results of optimizing a denoising problem with a first-order MRF and a higher-order Field of Experts MRF are shown in Figure 1.6. As expected, the more complicated higher-order model is better able to capture the local variations due to shading gradations and texture, and leads to a less “flattened” result. Of course, this complexity is not free — optimizing these higher-order models is more challenging. Consequently, we will use this particular denoising model as a benchmark for evaluating various higher-order optimization meth-

ods in later sections.

1.5.3 Curvature Regularizing Priors for Stereo

In some cases, certain image properties are simply impossible to express in a first-order MRF, as we will see from this example of stereo reconstruction. In the stereo reconstruction problem, we are given a left and right image taken by two cameras separated by a baseline. The cameras are calibrated so that rows in one image correspond to rows in the other image, and if a point in space maps to a pixel at column i of the right image, and column $i + \delta$ of the left image, then that point must be at a depth Δ/δ , where Δ is the separation of the cameras, known as the *baseline*. The difference δ is known as the disparity. We will turn stereo reconstruction into a labeling problem by discretizing the disparity into a finite set of pixel differences $\{0, 1, \dots, D\}$. Each variable x_i gives the disparity for each pixel i in the right image, from which we can infer the depth (given the baseline Δ).

We are interested in what kinds of priors are appropriate for stereo reconstruction. As with image denoising, we expect the depth image to consist of connected regions, so some sort of edge-preserving prior will be required. This prior should reflect the variation we expect within each region.

A simple prior is to put a cost on neighboring pixels whenever their disparities are different. There are other choices for this, but one possibility is the truncated L_1 cost that we used for denoising

$$f_{i,j}(x_i, x_j) = \min\{|x_i - x_j|, \tau\}. \quad (1.54)$$

This function has minimum cost when the disparity values for i and j are the

same — for this reason, this prior prefers the depth image to be composed of regions which are all at the same depth, i.e., regions which are each on a plane parallel to the camera image plane. For this reason, such priors are called *fronto-parallel priors*.

However, even simple depth images are not fronto-parallel: for example the depth image of a flat wall taken from a 45 degree angle is a slanting plane. Even with the simple assumption that real scenes are formed of totally flat objects, we would expect each region to be a slanting plane, but not necessarily fronto-parallel. An example of a prior based on this intuition (from the work of [102]) is to penalize the curvature of the depth map, as flat planes have no curvature. We do this by putting a cost on 3×1 windows of the image, with a cost function $f_{i,j,k}$ of the three disparities x_i, x_j and x_k (with corresponding depths $\frac{\Delta}{x_1}, \frac{\Delta}{x_2}, \frac{\Delta}{x_3}$):

$$f_{i,j,k}(x_i, x_j, x_k) = \min \left\{ \left| \frac{\Delta}{x_1} - 2\frac{\Delta}{x_j} + \frac{\Delta}{x_k} \right|, \tau \right\}. \quad (1.55)$$

The quantity $\frac{\Delta}{x_1} - 2\frac{\Delta}{x_j} + \frac{\Delta}{x_k}$ is a discrete approximation to the curvature of the depth map at pixel j , and as with the truncated L_1 cost, we cap this cost at τ to not overly penalize discontinuities between objects.

As shown in Figure 1.7, the results achieved by including a curvature regularizing prior are much more realistic, and do not arbitrarily chop the result into fronto-parallel planes, as in the first-order prior.

It is not just more natural to express curvature priors as a higher-order MRF, it is actually impossible to express these constraints using only neighboring pixel differences. To see this, in Figure 1.8, we have two possible sets of depths for a group of three pixels. The first possibility is planar, and should have low cost, while the other has a corner, and should be higher cost. A higher-order MRF with cliques of size 3 can distinguish between these two cases, and assign

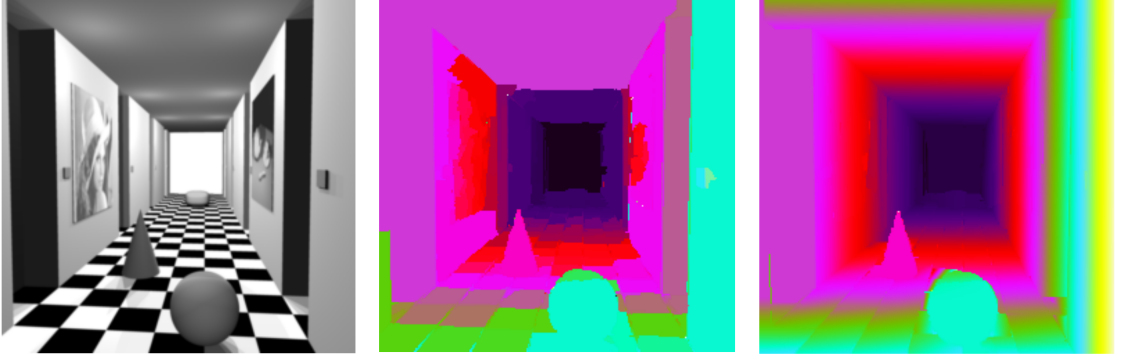


Figure 1.7: (Left) Example synthetic input to a stereo reconstruction problem. (Center) Reconstruction using a first-order prior, resulting in many fronto-parallel planes. (Right) Reconstruction using a third-order curvature regularizing prior, resulting in smooth planes and curves. All images from [102].

a higher cost to the case with a corner. However, a first-order MRF sees 4 pairs of pixels, each with the same distance $|x_i - x_j|$ in the disparities. By symmetry, we shouldn't penalize pairs of pixels sloping back left-to-right versus right-to-left, so we must assign the same cost to the plane and the corner.

Despite the extra (but necessary) expressiveness of these higher-order priors, they are challenging for existing optimization methods to handle. Ad-hoc methods have been proposed, including a reduction of these second-order cliques to a first-order model, as proposed in the paper which introduced this model [102]. However, it is our goal to show how to optimize such models in general. As with patch-based image denoising, this model is an important benchmark of higher-order inference algorithms.

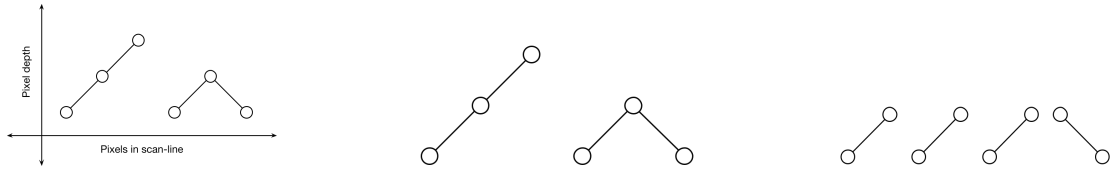


Figure 1.8: (Left) Two possible disparities for a set of three pixels. (Center) A higher-order model with size 3 cliques can see all three pixels, and assign different costs accordingly. (Right) A first-order model sees 4 different pairs of pixel differences, each with the same difference between the two pixels (with some slanting left to right, and some the opposite direction).

1.6 Conclusion

There are three main points from this introduction, which will hopefully frame the problem of inference in higher-order MRFs.

First, we have seen that MRFs arise from probabilistic inference, in particular in probabilistic inference problems in which the variables have spatial locality, as in the conditional dependence of nearby pixels in an image.

Not all MRFs are probabilistic inference problems, many are hand constructed without reference to forward models or prior distributions. However, all of the optimization techniques developed in further chapters will handle equations of the form

$$\min_{\mathbf{x}} \sum_C f_C(\mathbf{x}_C) \quad (1.56)$$

regardless of how the clique functions are chosen. That is, this equation is abstracts away the issues of probabilistic inference, to a form which can be directly handled by optimization algorithms.

Finally, we have seen how modeling more sophisticated priors leads to better answers, but can also result in harder inference problems. In particular, higher-

order models allow much greater flexibility for modeling than first-order priors, sometimes including constraints on the solution that cannot be expressed using only first-order terms.

CHAPTER 2

MATHEMATICAL BACKGROUND

In this chapter, we cover the mathematical tools necessary to present both the related work, as well as the main results of this thesis. To recall: our main problem is the MAP problem in MRFs, in which we are trying to minimize a function of the form

$$f(\mathbf{x}) = \sum_{C \in \mathcal{C}} f_C(\mathbf{x}_C) \quad (2.1)$$

where the variables $\mathbf{x} = (x_1, \dots, x_n)$ come from a label space $\mathcal{X} = \prod_i \mathcal{X}_i$, and the functions f_C are clique functions, each depending on a corresponding subset C of the variables, for each C in the clique set \mathcal{C} .

We will begin by looking at transformations of (2.1) called reparameterizations in Section 2.1, which will be a useful tool in many optimization algorithms. The special case of MRFs with binary labels in Section 2.2 provides most of the known theoretical results on hardness of optimization and approximation. Section 2.3 covers submodular functions, which are a class of MRFs for which exact optimization is tractable. Convex relaxations of the MRF optimization problem, called the Marginal Polytopes are presented in Section 2.4. The Marginal Polytope, and Linear Programming relaxations in general, form the basis for the main algorithms of this thesis, so we give an overview of Linear Programming in Section 2.5, along with the related concepts of convex functions (Section 2.6), linear programming duality (Section 2.7), and optimality conditions (Section 2.8). Finally, we give specific applications of these linear programming contexts to MRF inference, in particular the dual of the Local Marginal Polytope in Section 2.9. We conclude with a description of graph cuts algorithms, and how max-flow min-cut algorithms fit in to our linear programming framework

in Section 2.10.

2.1 Reparameterization

A simple but useful fact is that any particular function may have many ways of being written, and that different forms may reveal useful information about an optimization problem, or be otherwise more convenient for algorithms. These multiple representations of the same function are called reparameterizations.

Definition 18. Let $f, f' : \mathcal{X} \rightarrow \mathbb{R}$ be MRFs on the same label space, where

$$f(\mathbf{x}) = \sum_{C \in \mathcal{C}} f_C(\mathbf{x}_C) \quad f'(\mathbf{x}) = \sum_{C \in \mathcal{C}'} f'_C(\mathbf{x}_C). \quad (2.2)$$

Then f' is a reparameterization of f if $f'(\mathbf{x}) = f(\mathbf{x})$ for every $\mathbf{x} \in \mathcal{X}$. That is, they are equal as functions $\mathcal{X} \rightarrow \mathbb{R}$.

Note that the two functions f and f' may have very different values for the clique functions f_C and f'_C , and may even have different clique structures \mathcal{C} and \mathcal{C}' .

A common use of reparameterization is to put an MRF into a particular normal form as a starting point for optimization. For example, if we don't want to deal with negative values, we can always re-write an MRF so that each clique function is nonnegative (except for a constant term which may be negative).

Lemma 19. Any MRF $f : \mathcal{X} \rightarrow \mathbb{R}$ can be reparameterized to a form

$$f'(\mathbf{x}) = f'_0 + \sum_{C \in \mathcal{C}} f'_C(\mathbf{x}_C) \quad (2.3)$$

where for every $C \neq \emptyset$ we have $f'_C(\mathbf{x}_C) \geq 0$ for all $\mathbf{x}_C \in \mathcal{X}_C$.

Proof. For every $C \in \mathcal{C}$, let $\delta_C = \min_{\mathbf{x}_C} f_C(\mathbf{x}_C)$. We'll set $f'_C = f_C - \delta_C$, for $C \neq \emptyset$ and

$$f'_\emptyset = f_\emptyset + \sum_{C \neq \emptyset} \delta_C. \quad (2.4)$$

First, we have achieved our goal that the clique functions f'_C are nonnegative, since $f'_C(\mathbf{x}_C) = f_C(\mathbf{x}_C) - \delta_C$ and $f_C(\mathbf{x}_C) \geq \delta_C$ by choice of δ_C , hence $f'_C(\mathbf{x}_C) \geq 0$.

Now, to see that f' is a reparameterization of f , we just expand out the terms to see that we have added and subtracted constants in a way that cancels out. For any \mathbf{x} we have

$$\begin{aligned} f'(\mathbf{x}) &= f'_\emptyset + \sum_{C \neq \emptyset} f'_C(\mathbf{x}_C) \\ &= f_\emptyset + \sum_{C \neq \emptyset} \delta_C + \sum_{C \neq \emptyset} (f_C(\mathbf{x}_C) - \delta_C) \\ &= f_\emptyset + \sum_{C \neq \emptyset} f_C(\mathbf{x}_C) = f(\mathbf{x}) \end{aligned} \quad (2.5)$$

□

Most of the reparameterizations used later follow this same basic form: we take some of the cost from one clique function f_C and move it to another clique function $f_{C'}$, in such a way that the addition and subtraction balances out.

Even for the simple reparameterization in Lemma 19, we can prove useful facts about the original MRF f , in this case, giving an easy lower-bound on value of the optimal solution.

Corollary 20. *If f is reparameterized as in Lemma 19, then $f(\mathbf{x}) \geq f'_\emptyset$ for all $\mathbf{x} \in \mathcal{X}$. In particular, $\text{OPT}(f) \geq f'_\emptyset$.*

Proof. Since every term of f' has $f'_C(\mathbf{x}_C) \geq 0$ (except for possibly f'_\emptyset), we have $f(\mathbf{x}) = f'(\mathbf{x}) \geq f'_\emptyset$. □

Another useful reparameterization that moves cost from higher-order terms to unary terms is the *pencil* reparameterization, from the Min-sum diffusion algorithm of [100]. In the terminology of [100], a pencil¹ consists of a label a for a variable x_i , together with all the values $f_C(\mathbf{x}_C)$ of a single clique C , for just the labels \mathbf{x}_C with $x_i = a$. We can get a reparameterization by subtracting a value δ from all the $f_C(\mathbf{x}_C)$, and adding the same δ to the unary term $f_i(a)$.

$$\begin{aligned} f'_C(\mathbf{x}_C) &= \begin{cases} f_C(\mathbf{x}_C) - \delta & x_i = a \\ f_C(\mathbf{x}_C) & \text{otherwise} \end{cases} \\ f'_i(x_i) &= \begin{cases} f_i(x_i) + \delta & x_i = a \\ f_i(x_i) & \text{otherwise} \end{cases} \end{aligned} \tag{2.6}$$

This transformation doesn't change the cost: if \mathbf{x} happens to have $x_i = a$ then whatever the rest of the labels in \mathbf{x}_C are, we subtracted δ from $f_C(\mathbf{x}_C)$, while adding δ to $f_i(x_i)$, which cancel out. And if $x_i \neq a$ then we didn't change the value of f_C or f_i .

2.2 Pseudoboolean functions

In the following two sections, we will restrict our attention to binary problems; that is, MRFs where the label set for each variable is $\{0, 1\}$. We have already seen that binary MRFs are of particular importance for graph-cuts methods, since the min-cut problem is itself binary (each vertex is either on the s or t side of the cut). Consequently, such functions have attracted a good deal of research in the combinatorial optimization literature, as far back as [34], where they are known as *pseudoboolean functions*.

¹Presumably so-called because in the graph construction of [100] a pencil consists of edges from nodes representing all the $f_C(\mathbf{x}_C)$ to the single node representing x_i — these edges come to a point, thus resembling a pencil tip.

Definition 21. A pseudoboolean function (of n variables) is a function $f : \{0, 1\}^V \rightarrow \mathbb{R}$.

As with MRFs, many pseudoboolean functions have local structure in which they can be written as a sum of clique functions, so that $f(\mathbf{x}) = \sum_C f_C(\mathbf{x}_C)$. Each $f_C : \{0, 1\}^C \rightarrow \mathbb{R}$ is also a pseudoboolean function, defined just on the clique C . A pseudoboolean function f is higher-order if f has cliques C of size 3 or greater, and f is first-order if $|C| \leq 2$ for all cliques.

2.2.1 Representations of MRFs

For all MRFs (including multi-label MRFs), there are two main representations worth mentioning.

- In the *explicit* representation, the MRF f is given as a table of values for each clique function f_C . That is, for every $C \in \mathcal{C}$ and $\mathbf{x}_C \in \mathcal{X}_C$, the value $f_C(\mathbf{x}_C)$ is given as part of the input to the optimization algorithm.
- In the *implicit* (or *black box*) representation, the clique structure \mathcal{C} is given explicitly (as a collection of subsets $C \subseteq V$) but each clique function f_C is given as an oracle. That is, there is a piece of code or an algorithm which computes f_C given \mathbf{x}_C .

Generally, we assume that MRFs are given in the implicit representation, since this is the more general form², but both make sense in different scenarios. However, for the binary case, there are additional representations which make

²The black box is more general from the point of view of the algorithm, since a black box algorithm can be used on any other representation.

use of the combinatorial structure of binary labels: set functions and multilinear polynomials.

2.2.2 Set Functions

In the binary segmentation example (Section 1.3), we used a mapping between binary labelings $\{0, 1\}^V$ and subsets $S \subseteq V$ in order to convert between minimum cuts in a graph and segmentations. In general, we can use this conversion to map any pseudoboolean function $f : \{0, 1\}^V$ to a function taking subsets $S \subseteq V$ as input. Such functions are called set functions.

Definition 22. A set function *with base set* V is a function $f : 2^V \rightarrow \mathbb{R}$.

We can convert from boolean vectors $\mathbf{x} \in \{0, 1\}^V$ to subsets $S \subseteq V$ by defining

$$S(\mathbf{x}) = \{i \in V \mid x_i = 1\} \tag{2.7}$$

In the other direction, we get a boolean vector \mathbf{x} from a subset S by setting $x_i = 1$ if $i \in S$ and 0 otherwise. We'll denote this map $\mathbf{x}(S)$. It is easily checked that these maps are inverses of each other.

From this bijection, it is clear that we can just as easily define pseudoboolean functions to be functions $f : 2^V \rightarrow \mathbb{R}$, i.e., set functions. In many of the following results, it will be convenient to use either the boolean vector representation or the set function representation depending on the situation, so we will switch between them freely.

Finally, note that we have chosen the convention that $x_i = 1$ if and only if $i \in S$. The other choice of $x_i = 0$ if and only if $i \in S$ is also valid, but less

common.

2.2.3 Multilinear Polynomials

A distinguishing feature of the set $\{0, 1\}$ is that for both $x = 0$ and $x = 1$ we have $x^2 = x$. This lets us write polynomial functions of \mathbf{x} very simply: higher powers x_i^d collapse to linear terms x_i . For example, for $\mathbf{x} \in \{0, 1\}^V$ we have $7x_1^3x_3^2x_4^4 = 7x_1x_3x_4$. That is, all polynomials over $\{0, 1\}^V$ are multilinear.

Therefore, to specify a monomial, we don't need the powers on each variable, only the subset $H \subseteq V$ containing the variables in the monomial, and a coefficient a_H — in the example $7x_1^3x_3^2x_4^4$, we have $H = \{1, 3, 4\}$ and $a_H = 7$. With this notation, every monomial can be written $a_H \prod_{i \in H} x_i$ for some $H \subseteq V$ and $a_H \in \mathbb{R}$.

Definition 23. A multilinear polynomial is a pseudoboolean function of the form

$$f(\mathbf{x}) = \sum_{H \in \mathcal{H}} a_H \prod_{i \in H} x_i \quad (2.8)$$

where $\mathcal{H} \subseteq 2^V$ is the collection of variables in each monomial, and $a_H \in \mathbb{R}$ are the coefficients for each term.

2.2.4 Properties of Multilinear Polynomials

A non-obvious fact is that every pseudoboolean function can be written as a multilinear polynomial. In fact, this multilinear representation is unique, up to ignoring terms with coefficient $a_H = 0$. Since adding terms with $a_H = 0$ doesn't

change the polynomial, we get a canonical form by including all the terms for every $H \subseteq 2^V$ (setting $a_H = 0$ for $H \in 2^V \setminus \mathcal{H}$).

Lemma 24. *The multilinear polynomial representation of a pseudoboolean function f is unique. In particular, the coefficients a_H of f are given by*

$$a_H = \sum_{S \subseteq H} (-1)^{|H \setminus S|} f(\mathbf{x}(S)) \quad (2.9)$$

where $\mathbf{x}(S)$ is the binary vector corresponding to the set S (see Section 2.2.1).

We can simplify the proof of this lemma by noting the following formula for $f(\mathbf{x})$ in terms of the coefficients a_H :

Proposition 25. *For a multilinear polynomial f , we have*

$$f(\mathbf{x}(S)) = \sum_{H \subseteq S} a_H. \quad (2.10)$$

Proof. For $H \in \mathcal{H}$, we have two cases:

- If $H \subseteq S$, then in the binary vector $\mathbf{x}(S)$, $x_i = 1$ for all $i \in H$, so $a_H \prod_{i \in H} x_i = a_H$.
- If $H \not\subseteq S$ then there is some $j \in H$ with $j \notin S$. So, in $\mathbf{x}(S)$, we have $x_j = 0$, and hence $a_H \prod_{i \in S} x_i = 0$.

Therefore, we have that $a_H \prod_{i \in H} x_i = a_H \mathbb{I}[H \subseteq S]$, so

$$f(\mathbf{x}(S)) = \sum_{H \in \mathcal{H}} a_H \prod_{i \in H} x_i = \sum_{H \in \mathcal{H}} a_H \mathbb{I}[H \subseteq S] = \sum_{H \subseteq S} a_H \quad (2.11)$$

□

Using this fact, the lemma follows:

Proof of Lemma 24. We prove this by induction on $|H|$. For $H = \emptyset$, we have

$$f(\mathbf{x}(\emptyset)) = \sum_{H \subseteq \emptyset} a_H = a_\emptyset \quad (2.12)$$

so $a_\emptyset = f(\mathbf{x}(\emptyset)) = (-1)^{|\emptyset|} f(\mathbf{x}(\emptyset)) = \sum_{S \subseteq \emptyset} (-1)^{|\emptyset \setminus S|} f(\mathbf{x}(S))$.

Then, for general H we have

$$f(\mathbf{x}(H)) = \sum_{H' \subseteq H} a_{H'} = a_H + \sum_{H' \subsetneq H} a_{H'}. \quad (2.13)$$

By induction we expand out $a_{H'}$ to get

$$f(\mathbf{x}(H)) = a_H + \sum_{H' \subsetneq H} \sum_{S \subseteq H} (-1)^{|H' \setminus S|} f(\mathbf{x}(S)) \quad (2.14)$$

$$= a_H + \sum_{S \subseteq H} \sum_{H': S \subseteq H' \subsetneq H} (-1)^{|H' \setminus S|} f(\mathbf{x}(S)) \quad (2.15)$$

$$= a_H + \sum_{S \subseteq H} f(\mathbf{x}(S)) \sum_{H': S \subseteq H' \subsetneq H} (-1)^{|H' \setminus S|} \quad (2.16)$$

$$= a_H + \sum_{S \subsetneq H} -(-1)^{|H \setminus S|} f(\mathbf{x}(S)) \quad (2.17)$$

and rearranging, we have

$$a_H = f(\mathbf{x}(H)) + \sum_{S \subsetneq H} (-1)^{|H \setminus S|} f(\mathbf{x}(S)) \quad (2.18)$$

$$= \sum_{S \subseteq H} (-1)^{|H \setminus S|} f(\mathbf{x}(S)) \quad (2.19)$$

□

2.2.5 Computational Complexity and Hardness of Approximation

Pseudoboolean functions are closely linked with the family of NP-complete problems relating to boolean satisfiability. In particular, it is trivial to give a

reduction from the maximum satisfiability problem (MAX-SAT) to the pseudo-boolean optimization problem.

Recall that in boolean satisfiability problems, the literals (the boolean variables x_i , along with their negations \bar{x}_i) are combined into a set of logical formulas by the operators conjunction, \wedge , and disjunction, \vee . In the MAX-SAT problem, we have a set of *clauses*, each of which is a disjunction of a subset of the literals, for example, $C = x_1 \vee \bar{x}_3 \vee \bar{x}_9$. MAX-SAT is an optimization problem: our objective is to find a setting of the variables which maximizes the total number of satisfied clauses.

It is clear that this objective is already a pseudoboolean function: $f(\mathbf{x}) = |\{C \mid C \text{ is satisfied}\}|$ is a function $\mathbb{R}^n \rightarrow \mathbb{R}$. We can make this a minimization problem by minimizing $-f$, hence we have given a reduction from MAX-SAT to minimization of pseudoboolean functions, and therefore we have:

Theorem 26. *Minimization of pseudoboolean functions is NP-hard.*

Even restricting ourselves just to the simplest case of first-order pseudo-boolean functions doesn't make things any easier. Like MRFs, satisfiability problems are also distinguished by their order: MAX-2SAT restricts the size of all cliques to be of size at most 2, and is still NP-complete. In this case, the objective function f can be written as a first-order pseudoboolean function. Denote the set of clauses as C . For a clause C , we get a clique function³

$$f_C(x_i, x_j) = \begin{cases} 1 & C \text{ satisfied by } x_i, x_j \\ 0 & \text{otherwise} \end{cases} \quad (2.20)$$

³We are abusing notation to use C for both the clause and corresponding clique C , since both are just a collection of variables.

In this case, the MAX-2SAT objective is

$$f(\mathbf{x}) = \sum_{C \in \mathcal{C}} f_C(x_i, x_j). \quad (2.21)$$

Minimizing this is a first order pseudoboolean minimization problem, and therefore we have:

Theorem 27. *Minimizing first-order pseudoboolean functions is NP-hard.*

Having established that even simple pseudoboolean functions are NP-hard to optimize, we can ask the secondary question of whether any approximation algorithm is possible.

Definition 28. *For a number $\alpha \geq 1$, we say that a set of minimization problems $\{f\}$ can be α -approximated if there is a polynomial time algorithm which, for any instance f returns an assignment of the variables \mathbf{x} with $f(\mathbf{x}) \leq \alpha \text{OPT}(f)$. Such an algorithm is called an α -approximation for $\{f\}$.*

In other words, we may not be able to find an optimal solution \mathbf{x}^* in polynomial time, but we can at least find a solution \mathbf{x} with objective cost at most α times the optimal cost. Clearly, we want α as close to 1 as possible, so that the cost of our polynomial-time-computable solutions will be not far from the optimum.

For pseudoboolean functions, the question of approximability is somewhat complicated by the fact that the optimal value may be negative. We cannot have $f(\mathbf{x}) \leq \alpha \text{OPT}(f)$ for $\alpha > 1$ and $\text{OPT}(f) < 0$, since this would mean $f(\mathbf{x}) < \text{OPT}(f)$, which is impossible. So the above definition is meaningless for pseudoboolean functions which can be negative. To get around issues with negative objectives, we define \mathcal{P}^+ to be the set of all strictly positive pseudoboolean

functions, and \mathcal{P}_1^+ to be all the strictly positive first-order pseudoboolean functions.⁴ That is:

$$\begin{aligned}\mathcal{P}^+ &= \{f : \{0, 1\}^n \rightarrow \mathbb{R} \mid f(\mathbf{x}) > 0, \forall \mathbf{x} \in \{0, 1\}^n\} \\ \mathcal{P}_1^+ &= \{f : \{0, 1\}^n \rightarrow \mathbb{R} \mid f(\mathbf{x}) > 0, \forall \mathbf{x} \in \{0, 1\}^n, f \text{ is first order}\}\end{aligned}\tag{2.22}$$

Even with this assurance that the function is always positive, we still cannot approximately optimize first-order pseudoboolean functions.

Theorem 29. *There is no α -approximation for \mathcal{P}_1^+ unless $P = NP$.*

Proof. We will give a reduction from graph coloring, since we have the following theorem from [69].

Theorem 30 (Lund, Yannakakis). *There is an $\epsilon > 0$ such that graph-coloring cannot be n^ϵ -approximated, unless $P = NP$.*

Recall that in the graph-coloring problem, we are given a graph, and we must assign colors to each node such that no neighboring nodes share a color. The minimization problem for graph-coloring is to find the minimal number of colors for which there is a valid coloring.

Let $G = (V, E)$ be the given graph. Let $n = |V|$. Any graph is n colorable, since we can just give every node its own distinct color. So, colorings of G are functions $c : V \rightarrow \{1, \dots, n\}$ with $c(i) \neq c(j)$ for $\{i, j\} \in E$. The graph-coloring problem is to minimize the number of used colors:

$$\min_c |c(V)|\tag{2.23}$$

⁴Finding an algorithm to optimize \mathcal{P}^+ or \mathcal{P}_1^+ is known as a promise problem, since we are given the additional promise that the function is never negative.

We denote the minimum number colors used by a coloring as $\chi(c)$, and the minimum possible number of colors as $\chi(G)$.

First, we'll write graph coloring as a higher-order pseudoboolean function. Graph coloring is a multi-label problem (each node can take a label from $\{1, \dots, n\}$), however we can make it a binary problem by introducing variables $x_{i,j}$ where $x_{i,j} = 1$ if node i takes color j , and 0 otherwise. The binary vector \mathbf{x} defines a valid coloring if exactly one $x_{i,j} = 1$ for each i , and if whenever i, i' are neighbors we never have $x_{i,j} = x_{i',j} = 1$. The corresponding coloring c is $c(i) = j$ for the unique j with $x_{i,j} = 1$.

Then, the graph-coloring objective is a single higher-order term

$$f(\mathbf{x}) = \begin{cases} \chi(c) & \mathbf{x} \text{ gives a valid coloring } c \\ n + 1 & \text{otherwise} \end{cases} \quad (2.24)$$

Finally, we can reduce this higher-order pseudoboolean function to first order, using the results from later in this thesis (Chapter 4). In particular, there exists a function g (which we can compute in polynomial time) such that g is a function of \mathbf{x} and some auxiliary variables \mathbf{y} with $\min_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}) = f(\mathbf{x})$.

The minimum value of g is $\min_{\mathbf{x}, \mathbf{y}} g(\mathbf{x}, \mathbf{y}) = \min_{\mathbf{x}} f(\mathbf{x}) = \chi(G)$, the same as the minimum value to the original coloring problem. The function g is in \mathcal{P}_1^+ , so if there exists an α -approximation algorithm for this class, then it will compute a solution $(\mathbf{x}', \mathbf{y}')$ in polynomial time, with $g(\mathbf{x}', \mathbf{y}') \leq \alpha \chi(G)$.

Throwing away the auxiliary variables \mathbf{y}' , we can consider the cost of \mathbf{x}' alone. We have $f(\mathbf{x}') = \min_{\mathbf{y}} g(\mathbf{x}', \mathbf{y}) \leq g(\mathbf{x}', \mathbf{y}') \leq \alpha \chi(G)$. It's possible that \mathbf{x}' is not a valid coloring, but in this case, $f(\mathbf{x}') = n + 1$, so replace \mathbf{x} with the one corresponding to the coloring giving every node a distinct color (i.e., $c(i) = i$ for

$i = 1, \dots, n$), this can only reduce $f(\mathbf{x}')$ so now we have a valid coloring c with $\chi(c) = f(\mathbf{x}') \leq \alpha\chi(G)$, and thus we would have an α -approximation for graph coloring.

Therefore, an α -approximation algorithm for \mathcal{P}_1^+ would give an α -approximation algorithm for graph coloring, which we know cannot happen unless $P = NP$. \square

2.3 Submodular Functions

Despite the hardness of optimizing general pseudoboolean functions, there is an important subclass where we can efficiently do exact minimization. These functions, called *submodular*, are the basis for generalizing the min-cut problem to higher-order MRFs.

2.3.1 Decreasing Marginal Gains

Submodular functions are usually defined as set functions $f : 2^V \rightarrow \mathbb{R}$. Since we have already noted the equivalence between set functions and pseudoboolean functions (Section 2.2.2), all of these definitions will translate to pseudoboolean functions as well.

Intuitively, submodular functions capture the notion of diminishing marginal gains – specifically, diminishing marginal gains over a set of discrete binary choices.

As an example,⁵ consider the two (non-exclusive) choices of whether or not to have cake, and whether or not to have cookies. Having either cake or cookies is certainly better than having nothing, and having both cake and cookies is better than having either alone. However, having both is perhaps a bit too-much, and the added benefit of having “cookies with cake” over “just having cake”, isn’t as large as that of having cookies over nothing — that is, the marginal gain of adding cookies has decreased. An example table of utilities is in Table 2.1.

	No Cake	Cake
No Cookies	0	5
Cookies	3	7

Table 2.1: Utilities for various dessert options. The marginal gain for adding cookies to nothing is $3 - 0 = 3$, whereas the marginal gain of adding cookies to {Cake} is $7 - 5 = 2$.

To formalize this notion, we have a ground set V of choices, of which we may pick any subset. Each subset is assigned a value, or objective, $f(S) \in \mathbb{R}$. The marginal benefit of adding $i \in V$ to a subset $S \subseteq V$ is the difference in values $f(S \cup \{i\}) - f(S)$. To simplify notation, we will write $S \cup \{i\}$ as $S + i$, so this difference is $f(S + i) - f(S)$. To say we have decreasing marginal gain means that if we take a larger set T (where larger means $T \supseteq S$), then the marginal gain $f(T + i) - f(T)$ is smaller.

Definition 31. A function $f : 2^V \rightarrow \mathbb{R}$ is submodular if for every $S \subseteq T \subseteq V$, and $i \in V \setminus T$, we have

$$f(S + i) - f(S) \geq f(T + i) - f(T) \quad (2.25)$$

⁵Admittedly whimsical.

2.3.2 Equivalent Definitions of Submodularity

There are many different properties that are equivalent with Definition 31 which are useful depending on the situation.

Theorem 32. *The following are equivalent:*

1. f is submodular
2. for every $S, T \subseteq V$ we have

$$f(S \cup T) + f(S \cap T) \leq f(S) + f(T) \quad (2.26)$$

3. For every $S \subseteq V$ and $i, j \in V \setminus S$ with $i \neq j$ we have

$$f(S) + f(S + i + j) \leq f(S + i) + f(S + j) \quad (2.27)$$

Frequently, the second condition above is taken as the primary definition, however since they are equivalent, we could have taken any as the definition of submodular.

The third condition is notable for using many fewer constraints than the others (only $O(n^2 2^n)$ as opposed to $O(2^{2n})$ for (1) and (2)).

Proof. For (1) \Rightarrow (3), note that $S \subseteq S + j$, so from (2.25) we have

$$f(S + i) - f(S) \geq f(S + i + j) - f(S + j) \quad (2.28)$$

which can be rearranged to get (2.27) for every $S \subseteq V$ and $i, j \notin S$.

For (3) \Rightarrow (2), let S, T be any subsets of V . If $S \subseteq T$ then $S \cap T = S$ and $S \cup T = T$ so the inequality (2.26) trivially holds. The same is true if $T \subseteq S$, so we will now consider the case where $S \not\subseteq T$ and $T \not\subseteq S$.

Enumerate the elements of $T \setminus S$ as $T \setminus S = \{i_1, \dots, i_{k_1}\}$ and the elements of $S \setminus T$ as $S \setminus T = \{j_1, \dots, j_{k_2}\}$. Consider the following sum, over all pairs of $i_k, j_{k'}$

$$\begin{aligned} \sum_{k=1}^{k_1} \sum_{k'=1}^{k_2} [f(S + i_1 + \dots + i_k + j_1 + \dots + j_{k'}) - f(S + i_1 + \dots + i_{k-1} + j_1 + \dots + j_{k'}) \\ - f(S + i_1 + \dots + i_k + j_1 + \dots + j_{k'-1}) + f(S + i_1 + \dots + i_{k-1} + j_1 + \dots + j_{k'-1})] \end{aligned} \quad (2.29)$$

Simplify the above by letting $S_{k,k'} = S + i_1 + \dots + i_{k-1} + j_1 + \dots + j_{k'-1}$, and we get that (2.29) is

$$\sum_{k=1}^{k_1} \sum_{k'=1}^{k_2} [f(S_{k,k'} + i_k + j_{k'}) - f(S_{k,k'} + j_{k'}) - f(S_{k,k'} + i_k) + f(S_{k,k'})] \quad (2.30)$$

Each term of this sum is a rearrangement of (2.27), applied to $S_{k,k'}$, so the whole sum is ≤ 0 . We can split up this sum into 4 separate summations, and reindexing we get that (2.30) is equal to

$$\begin{aligned} \sum_{k=1}^{k_1} \sum_{k'=1}^{k_2} f(S_{k+1,k'+1}) - \sum_{k=1}^{k_1} \sum_{k'=1}^{k_2} f(S_{k+1,k'}) - \sum_{k=1}^{k_1} \sum_{k'=1}^{k_2} f(S_{k,k'+1}) + \sum_{k=1}^{k_1} \sum_{k'=1}^{k_2} f(S_{k,k'}) \\ = \sum_{k=2}^{k_1+1} \sum_{k'=2}^{k_2+1} f(S_{k,k'}) - \sum_{k=2}^{k_1+1} \sum_{k'=1}^{k_2} f(S_{k,k'}) - \sum_{k=1}^{k_1} \sum_{k'=2}^{k_2+1} f(S_{k,k'}) + \sum_{k=1}^{k_1} \sum_{k'=1}^{k_2} f(S_{k,k'}) \quad (2.31) \\ = f(S_{k_1+1,k_2+1}) - f(S_{k_1+1,1}) - f(S_{1,k_2+1}) + f(S_{1,1}) \\ = f(S \cup T) - f(S) - f(T) + f(S \cap T) \end{aligned}$$

Therefore, $f(S \cup T) + f(S \cap T) \leq f(S) + f(T)$ for all $S, T \subseteq V$, so we have that (3) \Rightarrow (2).

Finally, for (2) \Rightarrow (1), assume that (2.26) holds, and take any $S \subseteq T \subseteq V$ and $i \in V \setminus T$. Then $(S + i) \cup T = T + i$ and $(S + i) \cap T = S$, so we have

$$f(T + i) + f(S) = f((S + i) \cup T) + f((S + i) \cap T) \leq f(S + i) + f(T) \quad (2.32)$$

Rearranging, we get that (2.25) holds, so f is submodular. \square

2.3.3 Properties of Submodular Functions

Submodular functions are notable for sharing many properties with convex functions, and consequently they fill a similar role in discrete optimization problems as convex functions do for continuous optimization. See [68] for an excellent summary of the connections between convex and submodular functions.

The basic calculus of submodular functions is that they are closed under addition and multiplication by positive constants (but not subtraction or multiplication by negative constants).

Lemma 33. *Submodular functions are closed under positive linear combinations. That is, if f_1, \dots, f_k are submodular and $a_1, \dots, a_k \in \mathbb{R}$ are nonnegative, then $a_1 f_1 + \dots + a_k f_k$ is submodular.*

Proof. Since f_i is submodular and $a_i \geq 0$ we have $a_i f_i(S \cap T) + a_i f_i(T \cup T) \leq a_i f_i(S) + a_i f_i(T)$. Sum these inequalities together, and we get

$$\sum_i a_i f_i(S \cap T) + \sum_i a_i f_i(S \cup T) \leq \sum_i a_i f_i(S) + \sum_i a_i f_i(T) \quad (2.33)$$

□

A powerful tool in the analysis of convex functions are their subdifferentials: linear functions $\ell(x)$ which are also lower bounds, $\ell(x) \leq f(x)$. Submodular functions similarly have linear lower bounds, called *subbases*.

For binary labels, linear functions can be defined by a vector $\psi \in \mathbb{R}^V$. For each such ψ , we get a linear function (also called ψ by abuse of notation) with $\psi(S) = \sum_{i \in S} \psi_i$.

Definition 34. For a submodular function f , a subbase is a linear function ψ with $f(S) \geq \psi(S)$ for all $S \subseteq V$. A base of f is a subbase with $\psi(V) = f(V)$.

There is a simple algorithm to compute a base for any submodular function f , as long as $f(\emptyset) \geq 0$.⁶ In fact, we can greedily construct this vector ψ (this is known as Edmond's algorithm [16]). We let $\psi_1 = f(\{1\})$ and for $i = 2, \dots, n$ we set $\psi_i = f(\{1, \dots, i\}) - f(\{1, \dots, i-1\})$.

Lemma 35. The vector ψ defined above satisfies $f(S) \geq \psi(S)$ for all $S \subseteq V$ and $f(V) = \psi(V)$.

Proof. We prove the lemma inductively on the size of S . For $S = \emptyset$ we have $f(\emptyset) \geq \psi(\emptyset) = 0$.

For $S \neq \emptyset$ let i be the largest element of S . Since f is submodular, it has decreasing marginal gains, and $S - i \subseteq \{1, \dots, i-1\}$ so

$$\begin{aligned}
 f(S) &= f(S) - f(S - i) + f(S - i) \\
 &\geq f(\{1, \dots, i-1\} + i) - f(\{1, \dots, i-1\}) + f(S - i) \\
 &= \psi_i + f(S - i) \\
 &\geq \psi_i + \sum_{i \in S-i} \psi_i = \psi(S)
 \end{aligned} \tag{2.34}$$

For the second part of the claim, $\psi(V) = \sum_{i \in V} (f(\{1, \dots, i\}) - f(\{1, \dots, i-1\})) = f(V) - f(\emptyset) = f(V)$. \square

Submodular functions have additional structure, in addition to their similarity to convex functions. In particular, the set of all minimizers of a submodular function will be closed under intersections and unions. This follows easily from

⁶Note that if $f(\emptyset) < 0$ then there are no subbases of f at all, since $\psi(\emptyset) = 0$ by definition, and we require $f(\emptyset) \geq \psi(\emptyset)$.

one of the equivalent conditions for submodularity, Theorem 32. If S^* and T^* are both minimizers, then

$$f(S^* \cap T^*) + f(S^* \cup T^*) \leq f(S^*) + f(T^*) \quad (2.35)$$

and since S^* and T^* are minimizers, we must have $f(S^* \cap T^*) = f(S^* \cup T^*) = f(S^*)$.

In fact, we can generalize this to the sets where a submodular function is equal to a given subbase.

Definition 36. If ψ is a subbase of a submodular function f , the tight sets $\mathcal{T}(f, \psi)$ are all S for which $f(S) = \psi(S)$. That is, $\mathcal{T}(f, \psi) = \{S \subseteq V \mid f(S) = \psi(S)\}$.

Lemma 37. For a submodular function f and subbase ψ , $\mathcal{T}(f, \psi)$ is a lattice, meaning it is closed under intersection and union.

Proof. Let $S, T \in \mathcal{T}(f, \psi)$. So, in particular we have $f(S) = \psi(S)$ and $f(T) = \psi(T)$. We want to show that $S \cap T$ and $S \cup T$ are in $\mathcal{T}(f, \psi)$, so we want $f(S \cap T) = \psi(S \cap T)$ and $f(S \cup T) = \psi(S \cup T)$. Since $f \geq \psi$ we have $f(S \cap T) \geq \psi(S \cap T)$ and $f(S \cup T) \geq \psi(S \cup T)$. Now, since f is submodular, we have

$$\psi(S \cap T) + \psi(S \cup T) \leq f(S \cap T) + f(S \cup T) \leq f(S) + f(T) = \psi(S) + \psi(T) \quad (2.36)$$

Because ψ is linear, we have $\psi(S) + \psi(T) = \psi(S \cap T) + \psi(S \cup T)$, by inclusion-exclusion. Therefore, the inequalities in (2.36) are all equalities, and we have

$$f(S \cup T) + f(S \cap T) = \psi(S \cup T) + \psi(S \cap T) \quad (2.37)$$

Finally, since $f(S \cup T) \geq \psi(S \cup T)$ and $f(S \cap T) \geq \psi(S \cap T)$, we have that $f(S \cup T) = \psi(S \cup T)$ and $f(S \cap T) = \psi(S \cap T)$. \square

A particularly useful special case of the above lemma is when f is non-negative, meaning $f(S) \geq 0$ for all $S \subseteq V$. In this case, 0 is a sub-base of f , and the tight sets $\mathcal{T}(f, 0)$ are the zero-valued sets, $\mathcal{Z}(f) = \{S \subseteq V \mid f(S) = 0\}$.

Corollary 38. *If f is a non-negative submodular function, then the zero sets $\mathcal{Z}(f)$ form a lattice.*

2.3.4 Submodular First-order Pseudoboolean Functions

For checking if an arbitrary set function is submodular, the above definitions all have at least $O(n^2 2^n)$ equations to verify. There is some redundancy between the equations, but determining whether a function is submodular is NP hard in general [104]. For first-order pseudoboolean functions, whether or not a function is submodular is particularly easy to verify — all we need to check is that each of the pairwise coefficients (in the polynomial representation) is non-positive.

Lemma 39. *A first-order pseudoboolean function, given as a multilinear polynomial*

$$f(\mathbf{x}) = a_0 + \sum_i a_i x_i + \sum_{i,j} a_{i,j} x_i x_j \quad (2.38)$$

is submodular if and only if $a_{i,j} \leq 0$ for all $i, j \in V$.

Proof. Throughout this proof, we will treat f as a set function, with $f(S) := f(\mathbf{x}(S))$, where $\mathbf{x}(S)$ is the corresponding binary vector for the set S .

We use the explicit representation for the function values $f(S)$ given by Prop 25:

$$f(S) = \sum_{H \subseteq S} a_H. \quad (2.39)$$

We can relate this to one of our conditions for submodularity (Theorem 32) by expanding out $f(S) + f(S + i + j) - f(S + i) - f(S + j) \leq 0$:

$$0 \geq f(S) + f(S + i + j) - f(S + i) - f(S + j) \quad (2.40)$$

$$= \sum_{H \subseteq S} a_H + \sum_{H \subseteq S+i+j} a_H - \sum_{H \subseteq S+i} a_H - \sum_{H \subseteq S+j} a_H \quad (2.41)$$

$$= \left(\sum_{H \subseteq S+i+j} a_H - \sum_{H \subseteq S+j} a_H \right) - \left(\sum_{H \subseteq S+i} a_H - \sum_{H \subseteq S} a_H \right) \quad (2.42)$$

To complete the proof, we'll apply a simple proposition simplifying these sums over subsets:

Proposition 40. *For any set of coefficients a_H defined for subsetsets $H \subseteq V$ we have*

$$\sum_{H \subseteq S+i} a_H - \sum_{H \subseteq S} a_H = \sum_{H \subseteq S} a_{H+i} \quad (2.43)$$

To see this, note that that $2^{S+i} \setminus 2^S = \{S + i \mid S \in 2^S\}$. Using this fact, we have

$$f(S) + f(S + i + j) - f(S + i) - f(S + j) \quad (2.44)$$

$$= \sum_{H \subseteq S+j} a_{H+i} - \sum_{H \subseteq S} a_{H+i} \quad (2.45)$$

$$= \sum_{H \subseteq S} a_{H+i+j} = a_{i,j} \quad (2.46)$$

where the last line follows from $a_{H'} = 0$ for $|H'| > 2$ (because f is first-order). Therefore, we have that $f(S) + f(S + i + j) - f(S + i) - f(S + j) \leq 0$ if and only if $a_{i,j} \leq 0$, hence f is submodular if and only if $a_{i,j} \leq 0$ for all i, j . \square

Another way of recognizing first order submodular functions is if each individual pairwise term is submodular. First, note that we have that $f_{i,j}(x_i, x_j)$ is submodular if and only if a single inequality holds:

$$f_{i,j}(0, 0) + f_{i,j}(1, 1) \leq f_{i,j}(1, 0) + f_{i,j}(0, 1) \quad (2.47)$$

Then, since sums of submodular functions are submodular (Lemma 39) we have

Lemma 41. *A first-order pseudoboolean function*

$$f(\mathbf{x}) = \sum_i f_i(x_i) + \sum_{i,j} f_{i,j}(x_i, x_j) \quad (2.48)$$

is submodular if for each i, j we have $f_{i,j}$ is submodular (i.e., it satisfies (2.47)).

Note that this is a sufficient but not a necessary condition.

2.4 Local and Marginal Polytopes for MRFs

We will now begin to consider multilabel problems, i.e., those with more than just two labels. The next several sections are building to a key theoretical tool called the Local Marginal Polytope, which is a linear programming formulation of the MRF inference problem. In particular, we want to introduce this linear programming relaxation, as well as the major tools for dealing with linear programs, including duality and complementary slackness.

A major difficulty in optimizing MRFs is that they are discrete problems — there is a combinatorial space $\prod_i \mathcal{X}_i$ of possible states for \mathbf{x} , and as we have seen, it is difficult to make global statements about the function f (since efficiently finding either the minimum or any constant approximation to it would mean $P = NP$).

Contrast this with optimizing a convex, continuous function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. A major feature of such functions is that they always have a global optimum, and simple algorithms (including gradient descent) will always lead to this global

optimum. In this section we will show how MRF optimization can be cast as a particular kind of convex minimization problem called Linear Programming.

2.4.1 Weighted Averages as Linear Programs

To motivate Linear Programming, we will consider a simple example: finding the minimum element over a finite set $\{a_1, \dots, a_k\}$. The brute-force approach is of course to simply examine each element in turn, and remember the smallest. If we want to turn this into a continuous minimization problem instead, one way we can do this is to consider weighted averages of the elements a_i . That is, we have non-negative weights μ_i for each a_i with the weights summing to 1 (i.e., $\sum_i \mu_i = 1$) where the resulting weighted average, a_μ , is equal to the sum

$$a_\mu = \sum_i \mu_i a_i. \quad (2.49)$$

We know that the weighted average has to be between the minimum and maximum elements, so $a_\mu \geq \min\{a_i\}$. And, if we put all the weight on the minimizing i^* (i.e., $\mu_{i^*}^* = 1$ and $\mu_j^* = 0$ for $j \neq i^*$), then we have $a_{\mu^*} = a_{i^*} = \min\{a_i\}$.

In other words, the minimum over all weighted averages of the a_i is exactly the minimum element:

$$\min_{\mu: \mu \geq 0, \sum_i \mu_i = 1} \sum_i \mu_i a_i = \min_i \{a_i\} \quad (2.50)$$

This is a continuous optimization problem with a linear objective $\sum_i \mu_i a_i$, and linear constraints $\sum_i \mu_i = 1$, and $\mu_i \geq 0$ for $i = 1, \dots, n$, which makes (2.50) an example of a linear program.

2.4.2 Marginal polytopes

Recall that an MRF is called *discrete* if the state space \mathcal{X} is finite. Equivalently, each variable x_i has a finite label set $|\mathcal{X}_i| < \infty$.

Let $f(\mathbf{x}) = \sum_C f_C(\mathbf{x}_C)$ be a discrete MRF, with $\mathcal{X} = \prod_i \mathcal{X}_i$. For simplicity, we'll assume that $|\mathcal{X}_i| = \ell$ for all i , although nothing of the following breaks if we have non-uniformly-sized label sets.

Given the discussion in the previous section, we can convert this discrete, combinatorial optimization problem to a continuous linear program by minimizing the weighted average of all solutions. We have a weight for every state $\mathbf{x} \in \mathcal{X}$, which we will write as $\mu(\mathbf{x})$, and get a minimization problem

$$\begin{aligned} \min_{\mu} \quad & \sum_{\mathbf{x}} \mu(\mathbf{x}) f(\mathbf{x}) \\ \text{s.t.} \quad & \sum_{\mathbf{x}} \mu(\mathbf{x}) = 1 \\ & \mu \geq 0 \end{aligned} \tag{2.51}$$

Note that in the above, \mathbf{x} is no longer a free variable — we are instead summing the weights $\mu(\mathbf{x})$ over all possible states \mathbf{x} . In fact, if we treat the weighted average as instead giving a probability distribution over the states \mathbf{x} , then the objective $\sum_{\mathbf{x}} \mu(\mathbf{x}) f(\mathbf{x})$ is exactly the expectation of $f(\mathbf{x})$ when each state is chosen with probability $\mu(\mathbf{x})$. This explains our notation of $\mu(\cdot)$ as a probability density function on \mathcal{X} .

Thinking in terms of probability distributions gives us a pre-existing toolbox with which to reason about our problem. For example, it's clear that the expectation of f under the probability distribution μ is bounded between the minimum and maximum values of $f(\mathbf{x})$, and that if we're optimizing over all

probability distributions, the best we can do is to choose the minimizing \mathbf{x}^* with probability 1 (i.e., set $\mu(\mathbf{x}^*) = 1$ and $\mu(\mathbf{x}) = 0$ for $\mathbf{x} \neq \mathbf{x}^*$).

For any discrete set \mathcal{X} , let $\mathcal{P}[\mathcal{X}]$ be the set of all probability distributions on \mathcal{X} . That is, $\mathcal{P}[\mathcal{X}] = \{\mu \in \mathbb{R}^{\mathcal{X}} : \mu \geq 0, \sum_{\mathbf{x}} \mu(\mathbf{x}) = 1\}$. We will use $\langle \mu, f \rangle$ to denote the expectation of f with respect to μ , so that $\langle \mu, f \rangle = \sum_{\mathbf{x}} \mu(\mathbf{x}) f(\mathbf{x})$. With this notation, we get a very simple version of (2.51):

$$\min_{\mu \in \mathcal{P}[\mathcal{X}]} \langle \mu, f \rangle \quad (2.52)$$

A major drawback in optimizing over all probability distributions in $\mathcal{P}[\mathcal{X}]$ is that we have exploded the number of variables from $|V|$ to $\ell^{|V|}$. As a result, this linear program is much too large to solve efficiently even for small MRFs.

However, we have not yet used the clique structure of the MRF f . We can use the clique structure to get a much smaller number of variables — our eventual goal is to specify a probability distribution $\mu_C \in \mathcal{P}[\mathcal{X}_C]$ just for each clique C separately. For a clique of size k , this requires only ℓ^k variables, which is much smaller than the ℓ^n variables in $\mathcal{P}[\mathcal{X}]$ (assuming, per Remark 16, that $k \ll n$). Our first step towards this goal is to use linearity of expectation to expand out f in the objective of (2.52):

$$\begin{aligned} \langle \mu, f \rangle &= \left\langle \mu, \sum_C f_C \right\rangle \\ &= \sum_C \langle \mu, f_C \rangle \end{aligned} \quad (2.53)$$

Each of the terms $\langle \mu, f_C \rangle$ is an expectation of a function f_C which only depends on a few variables, namely the subset of variables in C . Therefore, we only care about the probability that a clique labeling \mathbf{x}_C is chosen. That is, we are interested in the probability that \mathbf{x} restricted to C is \mathbf{x}_C . We can compute this by

summing μ over all assignments of the remaining variables, $\mathbf{x}_{V \setminus C}$. To explain the notation, in the sum we get the combined vector $\mathbf{x} = (\mathbf{x}_C, \mathbf{x}_{V \setminus C})$. We will write $\mu|_C$ for the marginal distribution of μ on the subset of variables in C . This marginal probability is

$$\mu|_C(\mathbf{x}_C) = \sum_{\mathbf{x}_{V \setminus C}} \mu(\mathbf{x}) \quad (2.54)$$

A useful fact about marginalization is that it preserves expectations, as long as we marginalize onto the set of variables that a function depends on.

Proposition 42. *If $\mu \in \mathcal{P}[X]$ and f_C is a clique function, $f_C : X_C \rightarrow \mathbb{R}$, then $\langle \mu, f_C \rangle = \langle \mu|_C, f_C \rangle$.*

Proof. This equality is just a re-grouping of the sums in the definition of expectation:

$$\begin{aligned} \langle \mu, f_C \rangle &= \sum_{\mathbf{x}} \mu(\mathbf{x}) f_C(\mathbf{x}_C) \\ &= \sum_{\mathbf{x}_C} \sum_{\mathbf{x}_{V \setminus C}} \mu(\mathbf{x}) f_C(\mathbf{x}_C) \\ &= \sum_{\mathbf{x}_C} \mu|_C(\mathbf{x}_C) f_C(\mathbf{x}_C) = \langle \mu|_C, f_C \rangle \end{aligned} \quad (2.55)$$

□

This lets us re-write the expectation over μ in (2.52) as a sum over expectations on each clique:

$$\min_{\mu \in \mathcal{P}[X]} \sum_C \langle \mu|_C, f_C \rangle \quad (2.56)$$

This particular linear program is known as the *Marginal Polytope*, because the variables are marginal probabilities of the joint distribution μ . This problem is still equivalent to our original MRF optimization problem, but still has the problem of an exponentially large set of variables. We can fix the latter problem, but to do so, we must move to a relaxation of our original problem. Instead of

ensuring we have a global probability distribution μ , we will instead have a separate probability distribution μ_C on each clique C , and these distributions are required to only locally agree, meaning that if C and C' share a variable i , then they agree on the marginal distribution on that single variable: $\mu_C|_i = \mu_{C'}|_i$. We'll denote this single distribution for x_i by μ_i which we constrain to be equal to $\mu_C|_i$ for all C containing i .

$$\begin{aligned} \min_{\{\mu_C \in \mathcal{P}[\mathcal{X}_C]\}} \quad & \sum_C \langle \mu_C, f_C \rangle \\ \text{s.t.} \quad & \mu_C|_i = \mu_i \quad \forall C, i \in C \end{aligned} \tag{2.57}$$

This linear program is known as the *Local Marginal Polytope*, as we have replaced marginalization from a global joint probability distribution μ with local constraints on the consistency of the distributions μ_C .

This linear program is a great simplification compared to the full marginal polytope, and still provides a global lower-bound on the optimum of the original (integral) problem. However, we have moved to a relaxation, so it is possible that the optimal linear programming solution may not have any corresponding integral solution with as-good a value. In particular, whenever the clique functions are non-submodular, or whenever there are cycles in the graph, then the local marginal polytope may not be a tight relaxation.

2.5 Linear Programming

The marginal polytopes above are particular examples of a general class of problems called Linear Programs (LPs). A Linear Program is a constrained optimization problem, with variables $x_i \in \mathbb{R}$, a linear objective, and linear constraints. For

now, we will only consider the case where we have finitely many variables and constraints, although generalizations to infinitely many variables are possible.⁷

The constraints in a Linear Program may be equality constraints, or inequality constraints (either greater-or-equal or less-than-or-equal), and additionally, we may require that some of the variables are either non-negative or non-positive. As a result, without some unifying notation (which we will present shortly) there are many cases to consider to write down a “general LP”. As an example, the following LP with 3 variables and 3 constraints has each of these possibilities:

$$\begin{aligned}
& \min_{x_1, x_2, x_3} 3x_1 - 2x_2 + 7x_3 \\
& \text{s.t. } x_1 - x_2 = 3 \\
& \quad 2x_2 + 3x_3 \geq 2 \\
& \quad -3x_1 + x_3 \leq 4 \\
& \quad x_1 \geq 0 \\
& \quad x_2 \leq 0 \\
& \quad x_3 \in \mathbb{R}
\end{aligned} \tag{2.58}$$

The objective is linear in the 3 variables, as are each of the three constraints, with one each of an $=$, \geq and \leq constraint. Additionally, x_1 and x_2 are respectively required to be non-negative and non-positive, while x_3 may be positive or negative.

In general, an LP has m_1 greater-than constraints, and m_2 less-than constraints and m_3 equality constraints (with $m_1 + m_2 + m_3 = m$), as well as n_1 non-negative variables, n_2 non-positive variables, and n_3 variables which can be any real num-

⁷In particular, Linear Programming relaxations for continuous MRFs (where the label space is a continuous interval $[a, b] \subseteq \mathbb{R}$) are infinite dimensional. See the author’s paper [20] for an example of how to generalize the marginal polytope to continuous MRFs.

ber (with $n_1 + n_2 + n_3 = n$). Partitioning the indices by $I_1 = \{1, \dots, n_1\}$, $I_2 = \{n_1 + 1, \dots, n_1 + n_2\}$ and $I_3 = \{n_1 + n_2 + 1, \dots, n\}$ (and similarly for J_1, J_2, J_3) we get the general form of an LP:

$$\begin{aligned}
& \min_{\mathbf{x}} \sum_{i=1}^n c_i x_i \\
& \text{s.t.} \quad \sum_{i=1}^n a_{j,i} x_i \geq b_j \quad \forall j \in J_1 \\
& \quad \quad \sum_{i=1}^n a_{j,i} x_i \leq b_j \quad \forall j \in J_2 \\
& \quad \quad \sum_{i=1}^n a_{j,i} x_i = b_j \quad \forall j \in J_3 \\
& \quad \quad x_i \geq 0 \quad \forall i \in I_1 \\
& \quad \quad x_i \leq 0 \quad \forall i \in I_2
\end{aligned} \tag{2.59}$$

We can more easily organize these linear functions by writing them as dot-products and matrix-vector products: let $\mathbf{b} = (b_1, \dots, b_m)$ and $\mathbf{c} = (c_1, \dots, c_n)$ be vectors, and A be the $m \times n$ matrix with entries $a_{j,i}$. To handle the different types of constraints, let A_{J_k} be the submatrix of A with just the rows corresponding to J_k (for $k = 1, 2, 3$) and similarly \mathbf{b}_{J_k} the subvector of \mathbf{b} with rows from J_k . Then we can more compactly write (2.59) as

$$\begin{aligned}
& \min_{\mathbf{x}} \mathbf{c}^T \mathbf{x} \\
& \quad A_{J_1} \mathbf{x} \geq \mathbf{b}_{J_1} \\
& \quad A_{J_2} \mathbf{x} \leq \mathbf{b}_{J_2} \\
& \quad A_{J_3} \mathbf{x} = \mathbf{b}_{J_3} \\
& \quad \mathbf{x}_{I_1} \geq 0 \\
& \quad \mathbf{x}_{I_2} \leq 0
\end{aligned} \tag{2.60}$$

Note that inequalities regarding vectors (such as $\mathbf{x} \geq 0$) are always treated componentwise (i.e., $x_i \geq 0$ for all i).

2.5.1 Linear Cone Programming

Keeping track of which variables are positive or negative, and which constraints are equality vs. inequalities is tedious and can complicate equations. Simplifying this notational complexity gives us a good excuse to move to a slight generalization of Linear Programming called Cone Programming. Fortunately, the main theorems concerning Linear Programming (especially regarding duality) are most naturally stated using the theory of cones, so we will solve two problems at once by considering conic problems here. In Cone Programming, we require the variables and constraints to lie in a type of convex subset of \mathbb{R}^n called a cone.

Definition 43. A subset $K \subset \mathbb{R}^n$ is a cone if K is closed under addition and multiplication by nonnegative scalars $c \in \mathbb{R}, c \geq 0$. That is, for $\mathbf{x}, \mathbf{y} \in K$ we have $\mathbf{x} + \mathbf{y} \in K$ and for $c \geq 0$ we have $c\mathbf{x} \in K$.

The following 4 subsets of \mathbb{R} are particularly useful cones:

- $K_0 := \{0\}$
- $K_{\mathbb{R}} := \mathbb{R}$
- $K_{\geq} := \{x \in \mathbb{R} \mid x \geq 0\}$
- $K_{\leq} := \{x \in \mathbb{R} \mid x \leq 0\}$

It is trivial to verify that these are each closed under addition and multiplication by nonnegative scalars.

Using these cones, we can re-write the constraints of an LP: for example, the constraint $x_i \geq 0$ is the same as $x_i \in K_{\geq}$, and the constraint $\sum_i a_{j,i}x_i = b_j$ is the

same as $\sum_i a_{j,i}x_i - b_j \in K_0$. This lets us re-write all our constraints as membership in a cone. Given the partition (I_1, I_2, I_3) of the variables into $x_i \geq 0$, $x_i \leq 0$ and unconstrained x_i , we get a cone K defined by

$$K = \prod_{i \in I_1} K_{\geq} \times \prod_{i \in I_2} K_{\leq} \times \prod_{i \in I_3} K_{\mathbb{R}} \quad (2.61)$$

Then, the relation $\mathbf{x} \in K$ is identical to the intersection of the various constraints that $x_i \geq 0$ for $i \in I_1$, $x_i \leq 0$ for $i \in I_2$ and $x_i \in \mathbb{R}$ for $i \in I_3$.

Similarly, given the partition (J_1, J_2, J_3) of the constraints into \geq , \leq and $=$ relations, we can define

$$K' = \prod_{j \in J_1} K_{\geq} \times \prod_{j \in J_2} K_{\leq} \times \prod_{j \in J_3} K_0 \quad (2.62)$$

so that $A\mathbf{x} - \mathbf{b} \in K'$ is identical to the original constraints $A_{J_1}\mathbf{x} \geq \mathbf{b}_{J_1}$, $A_{J_2}\mathbf{x} \leq \mathbf{b}_{J_2}$ and $A_{J_3}\mathbf{x} = \mathbf{b}_{J_3}$.

Therefore, the general form of an LP (2.60) is much more simply expressed as

$$\begin{aligned} \min_{\mathbf{x}} \mathbf{c}^T \mathbf{x} \\ A\mathbf{x} - \mathbf{b} \in K' \\ \mathbf{x} \in K \end{aligned} \quad (2.63)$$

2.6 Convex Sets and Convex Functions

Much of the theory of Linear Programming comes from convex optimization — specifically, since the feasible set for an LP is a convex set, and the linear objective is likewise convex, an LP is an instance of a convex program. In this section, we will review the basics of convexity and convex programs, however, even the basics of convex optimization are large enough to fill a book [9].

The basic definitions of convexity is that any line joining two elements within a convex set is also contained in the set.

Definition 44. A set $\Omega \subseteq \mathbb{R}^n$ is convex if for every $a, b \in \Omega$ and $t \in [0, 1]$ we have $ta + (1 - t)b \in \Omega$.

For functions, convexity says that the set of points lying above the graph of f is convex. This set is called the epigraph.

Definition 45. A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex if for every $a, b \in \mathbb{R}^n$ and $t \in [0, 1]$ we have $f(ta + (1 - t)b) \leq tf(a) + (1 - t)f(b)$. Equivalently, f is convex if and only if the epigraph of f (i.e., the set $S \subseteq \mathbb{R}^{n+1}$, where $S = \{(\mathbf{x}, z) \mid z \geq f(\mathbf{x})\}$) is a convex set.

One of the most useful theorem regarding convex functions (for the purposes of optimization) is that all local minima of a convex function are also global optima.

Lemma 46. If $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is continuous, and \mathbf{x}^* is a local minimum of f (meaning $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in U$, where U is an open neighborhood of \mathbf{x}^*) then \mathbf{x}^* is a global minimum of f , meaning $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$.

Therefore, algorithms which converge to local optima (such as gradient descent methods) also always find a global optimum.

For proving results relating to convexity, one of the most powerful theorems is the hyperplane separation theorem. Recall that a hyperplane in \mathbb{R}^n is defined by an equation $v_1x_1 + \dots + v_nx_n = c$, or more compactly, $\mathbf{v}^T \mathbf{x} = c$. This divides \mathbb{R}^n in half, with those \mathbf{x} for which $\mathbf{v}^T \mathbf{x} \geq c$ on one side, and those with $\mathbf{v}^T \mathbf{x} < c$ on the other.

The hyperplane separation theorem says that for any closed convex set Ω , if we have any element \mathbf{x} outside Ω , then there is a hyperplane such that Ω is on one side of the hyperplane, and \mathbf{x} is on the other.

Theorem 47. *For any closed convex set $\Omega \subseteq \mathbb{R}^n$, and for any point \mathbf{x} not in Ω , there is a hyperplane separating \mathbf{x} from Ω — that is, there is a vector \mathbf{v} and scalar $c \in \mathbb{R}$ such that for any $\mathbf{y} \in \Omega$ we have $\mathbf{v}^T \mathbf{y} > c$ but $\mathbf{v}^T \mathbf{x} < c$.*

This theorem may not seem especially important, however in practice it holds a similar place to the Intermediate Value Theorem in one-dimensional calculus, in that many other more useful theorems follow directly from it.

Finally, we'll note that convexity immediately applies to our results of the previous section:

Lemma 48. *A cone $K \subseteq \mathbb{R}^n$ is convex.*

Proof. The cone K is closed under addition and multiplication by non-negative scalars, so let $a, b \in K$, and $t \in [0, 1]$. Then ta and $(1 - t)b$ are in K , and hence $ta + (1 - t)b \in K$ as well. □

2.7 Duality

One of the most important tools for working with linear programs is the theory of *duality*. Every linear program has another linear program associated to it, called the *dual program*. This dual program can be obtained by a purely mechanical process (which we will describe shortly), and provides a great deal

of information regarding solutions to our original linear program (henceforth called the *primal* problem).

At a high level, for a minimization problem (as in (2.60)) the dual program uses the constraints of the original problem to give a lower bound on the optimal objective. In fact, the dual program has a variable corresponding to every constraint of the primal program, and similarly a constraint corresponding to every variable of the primal. That is, dualization exchanges constraints and variables — this can be very helpful for problems with large numbers of constraints but few variables, or vice-versa.

2.7.1 Exchanging Minimization and Maximization

The simple observation at the heart of duality is that exchanging nested minimization and maximizations gives a lower bound.

Lemma 49. *If $f : X \times Y \rightarrow \mathbb{R}$ is any function (not necessarily continuous) and X, Y are any sets, then*

$$\min_{x \in X} \max_{y \in Y} f(x, y) \geq \max_{y \in Y} \min_{x \in X} f(x, y) \quad (2.64)$$

Proof. We'll define two helper functions, $g(x) = \max_{y \in Y} f(x, y)$ and $h(y) = \min_{x \in X} f(x, y)$. Note that, by definition, $\min_{x \in X} \max_{y \in Y} f(x, y) = \min_{x \in X} g(x)$, and similarly $\max_{y \in Y} \min_{x \in X} f(x, y) = \max_{y \in Y} h(y)$.

Let x be any element of X , and y any element of Y . Since $g(x)$ is the maximum over all y' of $f(x, y')$, we have $g(x) \geq f(x, y)$, and similarly since $h(y)$ is the minimum over all x' of $f(x', y)$ we have $h(y) \leq f(x, y)$. In particular, for any x and y

we always have

$$g(x) \geq f(x, y) \geq h(y) \quad (2.65)$$

A general fact regarding maxima and minima is that if we have two sets $A, B \subseteq \mathbb{R}$ with $a \geq b$ for any $a \in A$ and $b \in B$, then $\min A \geq \max B$. Therefore, we have

$$\min_{x \in X} \max_{y \in Y} f(x, y) = \min_{x \in X} g(x) \geq \max_{y \in Y} h(y) = \max_{y \in Y} \min_{x \in X} f(x, y) \quad (2.66)$$

□

2.7.2 Linear Programming Duality: An Example

The notion of duality just presented seems quite trivial — it is just a rule for interchanging minimization and maximization. However, in the context of Linear Programming, it becomes quite powerful. To see this in action, let's consider the following simple LP:

$$\begin{aligned} \min_{x_1, x_2} \quad & 3x_1 + 4x_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \geq 2 \\ & x_1, x_2 \geq 0 \end{aligned} \quad (2.67)$$

The feasible set of this LP is illustrated in Figure 2.1. We might guess that the minimizer of (2.67) is achieved at $(1, 0)$ with value 3. However, how can we prove this? With only 2 variables and 3 constraints, it's not especially difficult to prove that this is indeed the minimizer, for example by geometric arguments. With many variables and constraints, though, this becomes a much more difficult task.

If we can find some easily obtainable lower bound to the objective of (2.67), then it may be possible to prove something about the minimum value. In par-

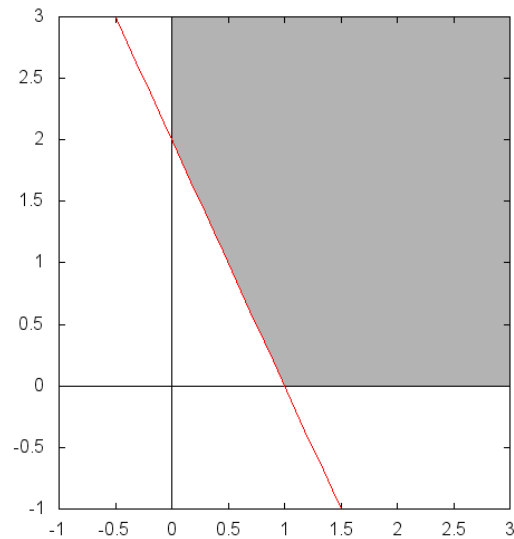


Figure 2.1: The feasible set of (2.67) is graphed above in grey.

ticular, if we could prove that the objective of (2.67) is always at least 3, then our proposed solution $(1, 0)$ with objective value 3 has to be the global minimizer (since any other solution can't have value less than 3). The duality lemma, Lemma 49, is our method to finding this lower bound.

The general recipe for constructing the dual is:

1. Take the constraints of the original LP and move them into the objective, using indicator functions (Section 1.2.2).
2. Write these indicator functions as a maximization over a new variable (called a Lagrange multiplier) times the original constraint.
3. Exchange minimization and maximization, using Lemma 49.
4. Take any terms that look like indicator functions out of the objective, and make them constraints.

Let's examine each of these steps in turn, for our example LP.

Re-writing constraints as indicator functions

Recall from Section 1.2.2 that we can convert any constrained optimization problem to an unconstrained problem by using indicator functions. If $g_j(\mathbf{x}) \geq 0$ is a constraint, then the indicator function in a minimization problem for g_j takes value ∞ for all infeasible solutions:

$$I_{g_j(\mathbf{x}) \geq 0}^{\min}(\mathbf{x}) = \begin{cases} 0 & g_j(\mathbf{x}) \geq 0 \\ \infty & \text{otherwise} \end{cases} \quad (2.68)$$

For a maximization problem, the indicator function takes value $-\infty$ for infeasible solutions:

$$I_{g_j(\mathbf{x}) \geq 0}^{\max}(\mathbf{x}) = \begin{cases} 0 & g_j(\mathbf{x}) \geq 0 \\ -\infty & \text{otherwise} \end{cases} \quad (2.69)$$

We can take any constrained optimization problem and get an equivalent problem by removing the constraint $g_j(\mathbf{x}) \geq 0$ and adding $I_{g_j(\mathbf{x}) \geq 0}(\mathbf{x})$ to the objective.

For our example problem, we'll use this to eliminate the constraint $2x_1 + x_2 \geq 2$. We get a new objective which is equal to the original objective, plus the indicator function for this constraint:

$$F(x_1, x_2) = 3x_1 + 4x_2 + I_{2x_1 + x_2 \geq 2}^{\min}(x_1, x_2) \quad (2.70)$$

As we've already noted, replacing constraints by indicator functions gives an unchanged optimization problem, so our original minimization (2.67) is equal to

$$\min_{x_1, x_2 \geq 0} 3x_1 + 4x_2 + I_{2x_1 + x_2 \geq 2}^{\min}(x_1, x_2) \quad (2.71)$$

Lagrange Multipliers

For the case of linear constraints, there is a simple way to re-write the indicator function which preserves the linear structure. To do so, let's look at what happens when we multiply the original constraints by a new, auxiliary variable (called a Lagrange Multiplier).

For the constraint $2x_1 + x_2 \geq 2$, we define the *residual* to be the quantity $2 - (2x_1 + x_2)$. When the residual is positive, then (x_1, x_2) is infeasible, and the residual is the amount by which the constraint has been violated. Conversely, when the residual is non-positive, then the constraint is satisfied.

Consider multiplying the residual by a non-negative variable $y \geq 0$:

$$y(2 - x_1 - x_2) \tag{2.72}$$

If the residual is positive, then we can send (2.72) to $+\infty$ by making y arbitrarily large. However, if the residual is non-positive then the biggest we can make (2.72) is 0, by setting $y = 0$. In other words, we have

$$\max_{y \geq 0} y(2 - 2x_1 - x_2) = \begin{cases} 0 & 2x_1 + x_2 \geq 1 \\ \infty & \text{otherwise} \end{cases} = I_{2x_1 + x_2 \geq 1}^{\min}(x_1, x_2) \tag{2.73}$$

Therefore, we have found a way to re-write the indicator function as a maximization over this Lagrange Multiplier. Applied to our example problem, we have that our original primal problem is equal to

$$\min_{x_1, x_2 \geq 0} 3x_1 + 4x_2 + \max_{y \geq 0} y(2 - 2x_1 - x_2) \tag{2.74}$$

Rearrange and exchange minimization and maximization

The next step of the process is purely algebraic. We want to exchange maximization and minimization, using Lemma 49, so we will group together all the terms containing x_1 and x_2 , and then apply our lemma.

$$\begin{aligned} & \min_{x_1, x_2 \geq 0} 3x_1 + 4x_2 + \max_{y \geq 0} y(2 - 2x_1 - x_2) \\ &= \min_{x_1, x_2 \geq 0} \max_{y \geq 0} x_1(3 - 2y) + x_2(4 - y) + 2y \\ &\geq \max_{y \geq 0} \min_{x_1, x_2 \geq 0} x_1(3 - 2y) + x_2(4 - y) + 2y \\ &= \max_{y \geq 0} 2y + \min_{x_1 \geq 0} [x_1(3 - 2y)] + \min_{x_2 \geq 0} [x_2(4 - y)] \end{aligned} \tag{2.75}$$

Transform indicator functions to constraints

Now we notice something interesting: the inner minimizations are a product of a variable x_i , times a linear function of the variable y . This is the exact same situation for our expression of the indicator function in (2.73), except now the roles of x_i and y have been reversed. In fact, we have

$$\begin{aligned} \min_{x_1 \geq 0} x_1(3 - 2y) &= \begin{cases} 0 & 2y \leq 3 \\ -\infty & \text{otherwise} \end{cases} \\ \min_{x_2 \geq 0} x_2(4 - y) &= \begin{cases} 0 & y \leq 4 \\ -\infty & \text{otherwise} \end{cases} \end{aligned} \tag{2.76}$$

These are the indicator functions for the constraints $2y \leq 3$ and $y \leq 4$, for the maximization problem over y . We can replace these indicator functions by the

corresponding constraints to get a linear program:

$$\begin{aligned} \max_{y \geq 0} \quad & 2y \\ \text{s.t.} \quad & 2y \leq 3 \\ & y \leq 4 \end{aligned} \tag{2.77}$$

This problem is called the *dual* of our original problem (2.67).

Obtaining a proof of optimality

We now return to our original question: how can we prove that the solution $(1, 0)$ with value 3 is optimal for the primal problem?

Note that the dual program (2.77) is a lower bound on our primal problem (2.67), because we exchanged minimization with maximization (and all other steps maintained equality).

Let's consider a potential dual solution: $y = \frac{3}{2}$. It's much easier to see that this is optimal for the dual problem because there's just a single variable, and we've raised it as high as possible without violating the constraint $2y \leq 3$. Furthermore, the value of this solution is 3.

Since the dual is a lower bound on the primal problem, we now know that any primal solution must have value at least 3 — therefore, our proposed solution of $(1, 0)$ is indeed optimal. The process by which we arrived at this proof was somewhat complicated; however, all the steps we performed were purely mechanical. Using the language of cone programming, we can give a simple recipe for obtaining the dual program of any LP.

2.7.3 Conic Duality

The example above shows how to obtain the dual LP for a single linear constraint. We'll now show how to find the dual for any Linear Program, using concepts from cone programming. The key definition to make this work is that of a *dual cone*.

Definition 50. For a cone $K \subseteq \mathbb{R}^n$, its dual cone K^* is the set of all $y \in \mathbb{R}^n$ whose dot product with all elements of K is nonnegative. So,

$$K^* := \{y \in \mathbb{R}^n \mid y^T x \geq 0, \forall x \in K\} \quad (2.78)$$

Using this definition, we can get a general formula for the dual of any linear cone program.

Theorem 51. The linear cone program

$$\begin{aligned} \min_x \quad & c^T x \\ & Ax - b \in K' \\ & x \in K \end{aligned} \quad (2.79)$$

has a dual program

$$\begin{aligned} \max_y \quad & b^T y \\ & A^T y - c \in -K^* \\ & y \in (K')^* \end{aligned} \quad (2.80)$$

In particular, weak duality holds, so that $\text{OPT}(2.79) \geq \text{OPT}(2.80)$.

Provided we can actually compute the dual cones K^* and $(K')^*$, then this gives a simple formula for computing the dual of any LP. Let's see how this works for the cones that define our basic equality and inequality constraints.

For $K_0, K_{\mathbb{R}}, K_{\geq}$ and K_{\leq} , we can compute their duals easily. Note that for $x, y \in \mathbb{R}$, $x^T y$ is just the product xy .

- $K_{\mathbb{R}}^* = K_0$: according to the definition, for y to be in $K_{\mathbb{R}}^*$ we would need $yx \geq 0$ for all $x \in \mathbb{R}$. Consider 3 cases for y : either $y > 0, y < 0$ or $y = 0$. If $y > 0$ set $x = -1$ (which has $x \in K_{\mathbb{R}}$). Therefore, we have $yx < 0$ for $x \in K_{\mathbb{R}}$ so $y \notin K_{\mathbb{R}}^*$. If $y < 0$ then set $x = 1$, and we have $yx < 0$ for $x \in K_{\mathbb{R}}$, and so $y \notin K_{\mathbb{R}}^*$. Finally, if $y = 0$ then $yx = 0$ for all x , so $y \in K_{\mathbb{R}}^*$.
- $K_0^* = K_{\mathbb{R}}$, since there is only one $x \in K_0$ and it is $x = 0$. Then, $y0 = 0$ for all $y \in \mathbb{R}$, and hence $K_0^* = \mathbb{R}$.
- $K_{\geq}^* = K_{\geq}$. We have two cases. If $y \geq 0$ then $yx \geq 0$ for any $x \geq 0$, and hence $y \in K_{\geq}^*$. If $y < 0$ then set $x = 1$ and we have $y \cdot 1 < 0$, and hence $y \notin K_{\geq}^*$.
- $K_{\leq}^* = K_{\leq}$ by similar argument.

When we have multiple constraints, the cone K is a cartesian product of cones, according to (2.62), so we need to know how taking the dual relates to cartesian products. Fortunately, the dual of a product is the product of the duals.

Proposition 52. *If K_1, \dots, K_n are cones, then $(K_1 \times \dots \times K_n)^* = K_1^* \times \dots \times K_n^*$.*

Proof. Let $K = K_1 \times \dots \times K_n$. To see that $K_1^* \times \dots \times K_n^*$ is the dual of K , let $\mathbf{x} \in K$. We'll write \mathbf{x} as the concatenation of subvectors $\mathbf{x}_i \in K_i$ so that $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$.

We know that for any i and $\mathbf{y}_i \in K_i^*$ that $\mathbf{y}_i^T \mathbf{x}_i \geq 0$, so in particular, letting $\mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_n)$ we have $\mathbf{y}^T \mathbf{x} \geq 0$. Therefore, $K_1^* \times \dots \times K_n^* \subseteq K^*$.

In the other direction, if $\mathbf{y} \in K^*$ then for each i , we want to show that $\mathbf{y}_i \in K_i^*$. Indeed, for any $\mathbf{x}_i \in K_i$ let \mathbf{x} be $(0, \dots, \mathbf{x}_i, \dots, 0)$, and we have $\mathbf{y}^T \mathbf{x} = \mathbf{y}_i^T \mathbf{x}_i$ must be ≥ 0 . This is true for any $\mathbf{x}_i \in K_i$, so $\mathbf{y}_i \in K_i^*$, and hence $K^* \subseteq K_1^* \times \dots \times K_n^*$. \square

In the remainder of the section, we prove Theorem 51. First, we can extend our observation about Lagrange variables resulting in indicator functions to the conic programming case.

Lemma 53. *If $\mathbf{x} \in K$ then $\max_{\mathbf{y} \in K^*} -\mathbf{y}^T \mathbf{x} = 0$. If $\mathbf{x} \notin K$, then $\max_{\mathbf{y} \in K^*} -\mathbf{y}^T \mathbf{x} = \infty$. That is,*

$$I_{\mathbf{x} \in K}^{\min}(\mathbf{x}) = \max_{\mathbf{y} \in K^*} -\mathbf{y}^T \mathbf{x} \quad (2.81)$$

Proof. If $\mathbf{x} \in K$, then by definition of K^* , since $\mathbf{y} \in K^*$ we have $\mathbf{y}^T \mathbf{x} \geq 0$ and hence $-\mathbf{y}^T \mathbf{x} \leq 0$. By choosing $\mathbf{y} = 0$ we get that the maximum of $-\mathbf{y}^T \mathbf{x}$ is equal to 0.

Now, let $\mathbf{x} \notin K$. Then, since K is closed and compact, by the Separating Hyperplane Theorem (Theorem 47) there exists $\tilde{\mathbf{y}}$ and c with $\tilde{\mathbf{y}}^T \mathbf{x} < c$ but $\tilde{\mathbf{y}}^T \mathbf{z} > c$ for all $\mathbf{z} \in K$. In particular, since $0 \in K$ we have $\tilde{\mathbf{y}}^T 0 > c$ so $c < 0$.

It turns out that this $\tilde{\mathbf{y}}$ defining the separating hyperplane is actually an element of the dual cone, $\tilde{\mathbf{y}} \in K^*$. To show this, assume by way of contradiction that there's some $\tilde{\mathbf{z}} \in K$ with $\tilde{\mathbf{y}}^T \tilde{\mathbf{z}} = \epsilon < 0$. Since K is a cone, and $\frac{\epsilon}{\epsilon} > 0$ we have $\frac{\epsilon}{\epsilon} \tilde{\mathbf{z}} \in K$. However, then $\tilde{\mathbf{y}}^T (\frac{\epsilon}{\epsilon} \tilde{\mathbf{z}}) = \frac{\epsilon}{\epsilon} \epsilon = c$ which contradicts $\tilde{\mathbf{y}}^T \mathbf{z} > c$ for all $\mathbf{z} \in K$. Therefore, there is no $\mathbf{z} \in K$ with $\tilde{\mathbf{y}}^T \mathbf{z} < 0$ and hence $\tilde{\mathbf{y}} \in K^*$.

Now, for our \mathbf{x} which is not in K we have that $\tilde{\mathbf{y}}^T \mathbf{x} < c$ so $-\tilde{\mathbf{y}}^T \mathbf{x} > -c > 0$. Since K^* is a cone, we have that $\lambda \tilde{\mathbf{y}} \in K^*$ for all $\lambda \geq 0$, so $-(\lambda \tilde{\mathbf{y}})^T \mathbf{x} > -\lambda c$ and hence

$$\max_{\mathbf{y} \in K^*} -\mathbf{y}^T \mathbf{x} \geq \max_{\lambda \geq 0} -(\lambda \tilde{\mathbf{y}})^T \mathbf{x} = \max_{\lambda \geq 0} -\lambda c = \infty \quad (2.82)$$

□

Note that this Lemma gives some intuition for separating hyperplanes, namely that they are directions along which we can send $\mathbf{y}^T \mathbf{x}$ to $-\infty$ for $\mathbf{x} \notin K$, and furthermore, separating hyperplanes of cones are elements of the dual cone.

Corollary 54.

$$I_{\mathbf{x} \in K}^{\max}(\mathbf{x}) = \min_{\mathbf{y} \in K^*} \mathbf{y}^T \mathbf{x} \quad (2.83)$$

Proof. This follows from $I_{\mathbf{x} \in K}^{\max}(\mathbf{x}) = -I_{\mathbf{x} \in K}^{\min}(\mathbf{x})$. \square

With this Lemma, we can prove the main theorem of this section.

Proof of Theorem 51. Let's start with our primal problem

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T \mathbf{x} \\ & A\mathbf{x} - \mathbf{b} \in K' \\ & \mathbf{x} \in K \end{aligned} \quad (2.84)$$

We then follow the recipe for our single variable example: (1) replace the constraint $A\mathbf{x} - \mathbf{b} \in K'$ with an indicator function, and then use Lemma 53 to write this as a maximization over a new variable \mathbf{y} . (2) Exchange maximization with minimization. (3) Replace terms that look like indicator functions with constraints:

$$\begin{aligned} \min (2.84) &= \min_{\mathbf{x} \in K} \mathbf{c}^T \mathbf{x} + I_{A\mathbf{x} - \mathbf{b} \in K'}^{\min}(\mathbf{x}) && \text{Replace constraints with indicators} \\ &= \min_{\mathbf{x} \in K} \mathbf{c}^T \mathbf{x} + \max_{\mathbf{y} \in (K')^*} -\mathbf{y}^T (A\mathbf{x} - \mathbf{b}) && \text{Lemma 53} \\ &= \min_{\mathbf{x} \in K} \max_{\mathbf{y} \in (K')^*} \mathbf{y}^T \mathbf{b} + \mathbf{c}^T \mathbf{x} - \mathbf{y}^T A\mathbf{x} \\ &\geq \max_{\mathbf{y} \in (K')^*} \min_{\mathbf{x} \in K} \mathbf{y}^T \mathbf{b} + (\mathbf{c} - A^T \mathbf{y})^T \mathbf{x} && \text{Exchange max with min} \\ &= \max_{\mathbf{y} \in (K')^*} \mathbf{y}^T \mathbf{b} + \min_{\mathbf{x} \in K} \mathbf{x}^T (\mathbf{c} - A^T \mathbf{y}) \\ &= \max_{\mathbf{y}} \{ \mathbf{b}^T \mathbf{y} \mid \mathbf{y} \in (K')^*, \mathbf{c} - A^T \mathbf{y} \in K^* \} && \text{Replace indicators with constraints} \end{aligned} \quad (2.85)$$

Finally, the last line is equal to our dual problem

$$\begin{aligned} \max_{\mathbf{y}} \quad & \mathbf{b}^T \mathbf{y} \\ & A^T \mathbf{y} - \mathbf{c} \in -K^* \\ & \mathbf{y} \in (K')^* \end{aligned} \tag{2.86}$$

□

2.8 Optimality for Linear Programs

The dual program is not just useful as a lower bound to the primal program — in fact, if we have a pair of solutions $(\mathbf{x}^*, \mathbf{y}^*)$ which are optimal for the primal and dual problems respectively, then these solutions obey a property called complementary slackness.

The main idea behind complementary slackness is that whenever a variable in the optimal solution x_i^* is nonzero, then the constraint corresponding to that variable in the dual program must be satisfied with equality (we say that such a constraint is *tight*). Conversely, whenever a dual constraint is not tight, the variable x_i must be 0, so complementary slackness is useful for proving facts about the sparseness of optimal solutions (i.e., that only certain variables x_i are nonzero).

Let's start with a linear conic program program, where we have expanded

out K and K' into a product of cones (as in (2.62)), with primal program

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T \mathbf{x} \\ & A\mathbf{x} - \mathbf{b} \in \prod_j K'_j \\ & \mathbf{x} \in \prod_i K_i \end{aligned} \tag{2.87}$$

and dual program

$$\begin{aligned} \max_{\mathbf{y}} \quad & \mathbf{b}^T \mathbf{y} \\ & A^T \mathbf{y} - \mathbf{c} \in -\prod_i K_i^* \\ & \mathbf{y} \in \prod_j (K'_j)^* \end{aligned} \tag{2.88}$$

For each constraint j , the *slack* is the value of the linear constraint, $(A\mathbf{x} - \mathbf{b})_j$.

Complementary slackness says that for each i , the product of the primal variable \mathbf{x}_i and the dual slack $(A^T \mathbf{y} - \mathbf{c})_i$ must be zero. Similarly, the product of the dual variable \mathbf{y}_j and the primal slack $(A\mathbf{x} - \mathbf{b})_j$ is also zero.

Theorem 55. *Given the linear conic programs above, if there exist optimal solutions \mathbf{x}^* and \mathbf{y}^* , then these satisfy $\mathbf{x}_i^T (A^T \mathbf{y} - \mathbf{c})_i = 0$ for all i and $\mathbf{y}_j^T (A\mathbf{x} - \mathbf{b})_j = 0$ for all j .*

We can specialize this to linear programs, where each cone K_i, K'_j are just subsets of \mathbb{R}^n .

Corollary 56. *If \mathbf{x}^* and \mathbf{y}^* are respectively primal and dual optimal for a Linear Program, and s_i, t_j are the primal and dual slack variables, given by $s_i = \mathbf{a}_i^T \mathbf{x} - b_i$, $t_j = \mathbf{y}^T \mathbf{a}_j - c_j$, then $x_i s_i = 0$ and $y_j t_j = 0$ for all i, j .*

Finally, we conclude with the Strong Duality theorem, which says that for LPs, we do not actually lose anything by exchanging maximization and minimization.

Theorem 57 (Strong Duality). *For a Linear Program, if both the primal and dual have feasible solutions, then their optimal values are equal.*

2.9 Duality for the Local Marginal Polytope

Recall the Local Marginal Polytope of Section 2.4

$$\begin{aligned}
& \min_{\{\mu_C\}} \sum_i \sum_{x_i} \mu_i(x_i) f_i(x_i) + \sum_C \sum_{\mathbf{x}_C} \mu_C(\mathbf{x}_C) f_C(\mathbf{x}_C) \\
& \text{s.t.} \quad \sum_{\mathbf{x}_{C \setminus i}} \mu_C(\mathbf{x}_C) = \mu_i(x_i) \quad \forall C, i \in C, x_i \in \mathcal{X}_i \\
& \quad \sum_{x_i} \mu_i(x_i) = 1 \quad \forall i \\
& \quad \mu \geq 0
\end{aligned} \tag{2.89}$$

Since we are interested in this linear program (as a relaxation of our MRF optimization problem), we should examine its dual, to see if there is any structure to the dual LP that can be exploited. As a bonus, we will also use this as an example of how to take the dual of an LP in practice.

The language of cones and dual cones are convenient for stating the main theorems of Linear Programming, but aren't necessarily the most convenient for doing calculations. However, the preceding sections have given us a recipe for computing the dual: (1) multiply the constraints by new Lagrange variables, and bring them into the objective as indicator functions, (2) exchange minimization with maximization, (3) group terms containing the original variables, and replace terms that look like indicator functions with constraints.

For the Local Marginal Polytope, we have two types of constraints, and correspondingly, two types of dual variables. The first set of constraints are for each

clique C , each $i \in C$ and each label $x_i \in \mathcal{X}_i$, so we'll denote the corresponding dual variable as $\lambda_{C,i}(x_i)$.⁸ The other constraints are for each i , with corresponding dual variable κ_i .

The indicator functions for these constraints are the max of the Lagrange variables times the residual, so we have $\lambda_{C,i}(x_i) \cdot (\mu_i(x_i) - \sum_{\mathbf{x}_C \setminus i} \mu_C(\mathbf{x}_C))$ for the first set of constraints and $\kappa_i (1 - \sum_{x_i} \mu_i(x_i))$ for the second set of constraints. Since these are equality constraints, these maximizations are over the dual cone $(K_0)^* = \mathbb{R}$. Therefore, we get that (2.89) is equal to

$$\min_{\mu \geq 0} \max_{\lambda, \kappa} \sum_i \sum_{x_i} \mu_i(x_i) f_i(x_i) + \sum_C \sum_{\mathbf{x}_C} \mu_C(\mathbf{x}_C) f_C(\mathbf{x}_C) \quad (2.90)$$

$$+ \sum_{C,i \in C} \sum_{x_i} \lambda_{C,i}(x_i) \left(\mu_i(x_i) - \sum_{\mathbf{x}_C \setminus i} \mu_C(\mathbf{x}_C) \right) + \sum_i \kappa_i \left(1 - \sum_{x_i} \mu_i(x_i) \right) \quad (2.91)$$

$$= \min_{\mu \geq 0} \max_{\lambda, \kappa} \sum_i \kappa_i + \sum_i \sum_{x_i} \mu_i(x_i) \left(f_i(x_i) - \kappa_i + \sum_{C:i \in C} \lambda_{C,i}(x_i) \right) \quad (2.92)$$

$$+ \sum_C \sum_{\mathbf{x}_C} \mu_C(\mathbf{x}_C) \left(f_C(\mathbf{x}_C) - \sum_{i \in C} \lambda_{C,i}(x_i) \right) \quad (2.93)$$

and hence

$$\geq \max_{\lambda, \kappa} \sum_i \kappa_i + \sum_C \sum_{\mathbf{x}_C} \min_{\mu_C(\mathbf{x}_C) \geq 0} \left[\mu_C(\mathbf{x}_C) \left(f_C(\mathbf{x}_C) - \sum_{i \in C} \lambda_{C,i}(x_i) \right) \right] \quad (2.94)$$

$$+ \sum_i \sum_{x_i} \min_{\mu_i(x_i) \geq 0} \left[\mu_i(x_i) \left(f_i(x_i) - \kappa_i + \sum_{C:i \in C} \lambda_{C,i}(x_i) \right) \right] \quad (2.95)$$

We have two expressions of the form $\min_{a \geq 0} a \cdot b$, which is $-\infty$ when $b < 0$ and 0 otherwise, so this is the same as the indicator for the constraint that $b \geq 0$.

Therefore, we can replace these expressions with the corresponding constraints

⁸We could have denoted this variable as λ_{C,i,x_i} , but as we'll see shortly, these variables have a natural interpretation as functions of x_i .

to get the linear program

$$\begin{aligned}
& \max_{\kappa, \lambda} \sum_i \kappa_i \\
& \text{s.t.} \quad \sum_i \lambda_{C,i}(x_i) \leq f_C(\mathbf{x}_C) \\
& \quad \kappa_i \leq f_i(x_i) + \sum_{C:i \in C} \lambda_{C,i}(x_i)
\end{aligned} \tag{2.96}$$

Note that we are maximizing over κ , and the only constraints involving κ are all of the form $\kappa_i \leq h_i(x_i)$ for some functions h_i . Specifically, let $h_i(x_i) = f_i(x_i) + \sum_{C:i \in C} \lambda_{C,i}(x_i)$, which we call the *height* of label x_i at variable i (following the language of [57]). Since we're maximizing over κ , we will always have $\kappa_i = \min_{x_i} h_i(x_i)$.

We can informally think of the dual variable $\lambda_{C,i}(x_i)$ as taking part of the cost $f_C(x_C)$, and redistributing it to the unary terms. The height functions $h_i(x_i)$ can be thought of as the original cost $f_i(x_i)$, plus any redistribution $\lambda_{C,i}$ from the cliques to the unary terms at i . The dual is always a lower bound on the value $f(\mathbf{x})$ of any labeling.

2.10 First-order Binary MRFs and Minimum Cut

We will conclude the chapter by returning to binary first-order problems, and in particular how we can use max-flow/min-cut to solve them. First, we will give a general solution for solving any submodular first-order MRF with min-cut. Furthermore, we will also show how the Local Marginal Polytope for these problems is itself equivalent to a cut polytope — that is, max-flow/min-cut is not just a convenient algorithm for this purpose, but actually is the same prob-

lem from a Linear Programming perspective.

2.10.1 Solving First-order Submodular MRFs with Graph Cuts

We will start with a binary first-order submodular MRF, that is, a minimization problem of the form

$$\min_{\mathbf{x}} \sum_i f_i(x_i) + \sum_{i,j} f_{i,j}(x_i, x_j) \quad (2.97)$$

where the variables x_i are in $\{0, 1\}$ and each $f_{i,j}$ is submodular, meaning it satisfies

$$f_{i,j}(0, 0) + f_{i,j}(1, 1) \leq f_{i,j}(1, 0) + f_{i,j}(0, 1). \quad (2.98)$$

In computer vision, Graph Cuts refers to the various different algorithms which solve MRF minimization problems using the minimum-cut/maximum-flow problem. For the simplest case of first-order submodular MRFs, we can reduce (2.97) directly to a minimum cut problem.

To do so, it is easiest to first reparameterize the problem. We have already seen in Section 2.1 that there are several ways to change the values of the f_i and $f_{i,j}$ while keeping the total energy function the same for any labeling \mathbf{x} . In this case, we want to find a reparameterization of (2.97) so that $f_{i,j}(x_i, x_j) \geq 0$ for all $x_i, x_j \in \{0, 1\}$, and so that $f_{i,j}(0, 0) = f_{i,j}(0, 1) = f_{i,j}(1, 1) = 0$.⁹ I.e., f_i is nonzero for a single label $f(1, 0)$.

We use a series of reparameterizations to make this happen. Then, for each $f_{i,j}$ we first subtract $f_{i,j}(0, 0)$ from all values $f_{i,j}(x_i, x_j)$, and add the corresponding amount to the constant term of f . Next, we use the pencil reparameteri-

⁹Note that we've broken the symmetry between i and j with these requirements, so we'll assume that each unordered edge $\{i, j\}$ has $i < j$.

zation (2.6) of Section 2.1 to subtract $\delta = (f_{i,j}(1, 1) - f_{i,j}(0, 1))$ from $f_{i,j}(1, x_j)$ for $x_j \in \{0, 1\}$ and add δ to $f_i(1)$. Similarly, we subtract $\delta' = (f_{i,j}(0, 1) - f_{i,j}(0, 0))$ from $f_{i,j}(x_i, 1)$ for $x_i \in \{0, 1\}$ and add δ' to $f_j(1)$.

Overall, we have a new energy function $f'_{i,j}$ given by:

$$\begin{aligned}
f'_{i,j}(0, 0) &= f_{i,j}(0, 0) - f_{i,j}(0, 0) = 0 \\
f'_{i,j}(0, 1) &= f_{i,j}(0, 1) - f_{i,j}(0, 0) - (f_{i,j}(0, 1) - f_{i,j}(0, 0)) = 0 \\
f'_{i,j}(1, 0) &= f_{i,j}(1, 0) - f_{i,j}(0, 0) - (f_{i,j}(1, 1) - f_{i,j}(0, 1)) \\
f'_{i,j}(1, 1) &= f_{i,j}(1, 1) - f_{i,j}(0, 0) - (f_{i,j}(1, 1) - f_{i,j}(0, 1)) - (f_{i,j}(0, 1) - f_{i,j}(0, 0)) = 0
\end{aligned} \tag{2.99}$$

Note that, because $f_{i,j}$ is submodular, we have that $f'_{i,j}(1, 0) \geq 0$, as the expression above for $f'_{i,j}(1, 0)$ is just a rearrangement of the equation defining submodularity (2.98).

For the unary terms, we sum up the added δ for each edge i, j to get

$$f'_i(1) = f_i(1) + \sum_{j:i < j} f_{i,j}(1, 1) - f_{i,j}(0, 1) + \sum_{j:j < i} f_{i,j}(1, 0) + f_{i,j}(0, 0) \tag{2.100}$$

Now, recall that in a minimum cut problem, we have a graph with a source s and sink t , and vertices V , and for each edge e we have a capacity c_e . A cut $S \subseteq V + s + t$ has $s \in S$ but $t \notin S$. A directed edge (u, v) is *cut* if $u \in S$ but $v \notin S$, and the total cost of a cut S is the sum of capacities of all cut edges.

In particular, if we identify cuts S with binary labels \mathbf{x} where $x_i = 1$ if and only if $i \in S$, then an edge (i, j) is cut if and only if $x_i = 1$ and $x_j = 0$, and hence only if we would pay the cost $f'_{i,j}(1, 0)$ in the objective f' .

Therefore, it's easy to construct a flow network from f' : for each undirected edge $\{i, j\}$ with $i < j$, we add a directed edge (i, j) with capacity $f'_{i,j}(1, 0)$. We

also add edges between s, t and the original nodes $i \in V$ according to the same construction we used in our binary segmentation example of Section 1.3: $c_{s,i} = f'_i(0)$ and $c_{i,t} = f'_i(1)$.

Using the bijection between binary labels \mathbf{x} and cuts S , we have that the cost of a cut is exactly equal to $f'(\mathbf{x})$, and hence we have reduced the MRF inference problem to the min-cut problem in a graph.

2.10.2 Linear Programs for Min-Cut

We can get a Linear Program for minimum cut in a graph by first starting with an integer programming formulation of the problem. This integer program has a set of binary variables $p_i \in \{0, 1\}$ which are 1 whenever i is in the cut S and 0 otherwise. For each edge, we will have an auxiliary variable $a_{i,j}$ which is 1 if the edge (i, j) is cut. We can enforce that $a_{i,j}$ behaves appropriately by adding the constraint $a_{i,j} \geq p_i - p_j$. Note that whenever $p_i = 1$ and $p_j = 0$ then the edge (i, j) is cut, and this constraint forces $a_{i,j} \geq 1 - 0 = 1$; for all other values of p_i, p_j though, we can set $a_{i,j}$ to 0 and still satisfy the constraint.

Therefore, we have an integer programming version of the minimum cut problem:

$$\begin{aligned}
& \min_{p, a} \sum_i c_{i,t} p_i + \sum_i c_{s,i} (1 - p_i) + \sum_{i,j} a_{i,j} c_{i,j} \\
& \text{s.t. } a_{i,j} \geq p_i - p_j \\
& \quad a_{i,j} \geq 0 \\
& \quad p_i \in \{0, 1\}
\end{aligned} \tag{2.101}$$

We get a linear program by relaxing the integrality constraint $p_i \in \{0, 1\}$ to

the linear constraint $p_i \in [0, 1]$.

2.10.3 Local Marginal Polytope for First-order Binary Problems

Let's consider the Local Marginal Polytope, in the case where we have only first order clique functions, $f_{i,j}$, and the labels are binary. In this section, we'll show that this LP is exactly equal to a min-cut LP, in the case where all the $f_{i,j}$ are submodular. To start, let's look at just the part of the Local Marginal Polytope dealing with just a single edge $\{i, j\}$ and all objective terms and constraints that deal with the variables $\mu_{i,j}$. This is

$$\begin{aligned}
& \min_{\mu_{i,j}} \langle \mu_{i,j}, f_{i,j} \rangle \\
& \text{s.t. } \sum_{x_i} \mu_{i,j}(x_i, x_j) = \mu_j(x_j) \quad \forall x_j \\
& \quad \sum_{x_j} \mu_{i,j}(x_i, x_j) = \mu_i(x_i) \quad \forall x_i \\
& \quad \mu_{i,j} \geq 0
\end{aligned} \tag{2.102}$$

We'll expand out all the sums over labels to be totally explicit:

$$\begin{aligned}
& \min_{\mu_{i,j}} \mu_{i,j}(0,0)f_{i,j}(0,0) + \mu_{i,j}(0,1)f_{i,j}(0,1) \\
& \quad + \mu_{i,j}(1,1)f_{i,j}(1,0) + \mu_{i,j}(1,1)f_{i,j}(1,1) \\
& \text{s.t. } \mu_{i,j}(0,0) + \mu_{i,j}(0,1) = \mu_i(0) \\
& \quad \mu_{i,j}(1,0) + \mu_{i,j}(1,1) = \mu_i(1) \\
& \quad \mu_{i,j}(0,0) + \mu_{i,j}(1,0) = \mu_j(0) \\
& \quad \mu_{i,j}(0,1) + \mu_{i,j}(1,1) = \mu_j(1) \\
& \quad \mu_{i,j}(0,0), \mu_{i,j}(0,1), \mu_{i,j}(1,0), \mu_{i,j}(1,1) \geq 0
\end{aligned} \tag{2.103}$$

This subset of the LP has 4 variables and 4 equality constraints. Just counting constraints, we'd expect there to be just a single value of $\mu_{i,j}$ satisfying these constraints (the solution of the 4×4 linear system given by these equality constraints). However, the constraints are not linearly independent (for example, add the first two and subtract the second two to get $0 = 0$), so there are only 3 independent constraints. Thus, there is a single variable which parameterizes solutions to this linear system.

We'll let this single parameter be $a_{i,j}$, which we'll (somewhat arbitrarily, for now) set to $a_{i,j} = \mu_{i,j}(1, 0)$. To simplify notation, let $p_i = \mu_i(1)$, $p_j = \mu_j(1)$. Note that $\mu_i(0) = 1 - p_i$ and $\mu_j(0) = 1 - p_j$. Then, manipulating the above equalities we have

$$\begin{aligned}\mu_{i,j}(1, 1) &= \mu_i(1) - \mu_{i,j}(1, 0) = p_i - a_{i,j} \\ \mu_{i,j}(0, 0) &= \mu_j(0) - \mu_{i,j}(1, 0) = 1 - p_j - a_{i,j} \\ \mu_{i,j}(0, 1) &= \mu_j(1) - \mu_{i,j}(1, 1) = p_j - p_i + a_{i,j}\end{aligned}\tag{2.104}$$

From the positivity constraints on the $\mu_{i,j}$ we get 4 inequalities: $a_{i,j} \geq 0$, $p_i - a_{i,j} \geq 0$, $1 - p_j - a_{i,j} \geq 0$ and $p_j - p_i + a_{i,j} \geq 0$. Additionally, the objective is a linear equation in $a_{i,j}$ and p_i, p_j :

$$\begin{aligned}\langle \mu_{i,j}, f_{i,j} \rangle &= (1 - p_j - a_{i,j})f_{i,j}(0, 0) + (p_j - p_i + a_{i,j})f_{i,j}(0, 1) + a_{i,j}f_{i,j}(1, 0) + (p_i - a_{i,j})f_{i,j}(1, 1) \\ &= \left(-f_{i,j}(0, 0) + f_{i,j}(0, 1) + f_{i,j}(1, 0) - f_{i,j}(1, 1)\right)a_{i,j} \\ &\quad + \left(f_{i,j}(1, 1) - f_{i,j}(0, 1)\right)p_i + \left(f_{i,j}(0, 1) - f_{i,j}(0, 0)\right)p_j + f_{i,j}(0, 0)\end{aligned}\tag{2.105}$$

We'll write the coefficient of a as $\delta_{i,j} := -f_{i,j}(0, 0) + f_{i,j}(0, 1) + f_{i,j}(1, 0) - f_{i,j}(1, 1)$. Note that $\delta_{i,j}$ is exactly the equation defining whether $f_{i,j}$ is submodular, so $f_{i,j}$ is submodular if and only if $\delta_{i,j} \geq 0$.

We have assumed that $f_{i,j}$ is submodular, so $\delta_{i,j} \geq 0$. Since we are minimizing

the objective, we want $a\delta_{i,j}$ to be minimized, and hence a should be as small as possible. Thus, the constraints that matter are $a_{i,j} \geq 0$ and $p_j - p_i + a_{i,j} \geq 0$.

We can therefore take every term and replace the four variables $\mu_{i,j}(0,0), \dots, \mu_{i,j}(1,1)$ with the single variable $a_{i,j}$ and the constraints $a_{i,j} \geq 0, a_{i,j} \geq p_i - p_j$. For each pairwise term, we get terms in the objective for $\delta_{i,j}a_{i,j}$ as well as $(f_{i,j}(1,1) - f_{i,j}(0,1))p_i$ and $(f_{i,j}(0,1) - f_{i,j}(0,0))p_j$. Collecting all the terms for the p_i across all pairwise terms, together with the original value of $(f_i(1) - f_i(0))p_i$, we get a linear term

$$\left(f_i(1) - f_i(0) + \sum_{j \in N(i)} (f_{i,j}(1,1) - f_{i,j}(0,1) + f_{i,j}(1,0) - f_{i,j}(0,0)) \right) p_i \quad (2.106)$$

Letting δ_i be the coefficient of p_i in the above equation, and we get a linear program:

$$\begin{aligned} \min_{p,a} \quad & \sum_i \delta_i p_i + \sum_{i,j} \delta_{i,j} a_{i,j} \\ \text{s.t.} \quad & a_{i,j} \geq p_i - p_j \\ & a_{i,j} \geq 0 \\ & 0 \leq p_i \leq 1 \end{aligned} \quad (2.107)$$

Finally, note that this is exactly the same linear program we derived for the minimum-cut problem, under the same reparameterization which sets $f_{i,j}(0,0) = f_{i,j}(0,1) = f_{i,j}(1,1) = 0$.

CHAPTER 3

RELATED WORK

There is considerable existing literature on higher-order MRFs — as we have seen, MRFs are a useful framework for probabilistic inference in computer vision, and higher-order MRFs allow more expressiveness for modeling priors that reflect the properties of real images. For an extensive survey on MRFs (both higher-order and first-order) in computer vision, see [97]. Here, we will cover in detail previous methods most relevant to optimizing higher-order MRFs.

In Section 3.1, we’ll examine the various examples using higher-order MRFs to model computer vision problems. In the rest of the chapter, we’ll break down inference algorithms into a rough categorization. As seen in Section 2.2, binary MRFs have special structure, and are closely related to combinatorial problems like min-cut. Consequently, there is a large diversity of approaches for this special case, which we’ll look at in Section 3.2.

For multi-label MRFs, we’ll categorize algorithms by how they use the Marginal Polytope of Section 2.4. Methods which either do not use Linear Programming at all (or which use only the primal LP) we’ll call Primal methods. Many of these are move-making algorithms, meaning they are local-search algorithms in the space of labels \mathcal{X} . Methods using the dual LP are largely message passing algorithms — they optimize the dual program iteratively by performing local updates around each variable, and then sending messages to their neighbors. Finally, Primal-Dual algorithms maintain both a primal and dual solution, and use the information contained in the dual to significantly accelerate the convergence of the primal solution. We will consider Primal, Dual and Primal-Dual methods in Sections 3.3, 3.4 and 3.5 respectively.

3.1 Higher-Order Models in Computer Vision

Higher-order MRFs have become increasingly important in computer vision, as they can incorporate sophisticated priors which better reflect natural images than first-order models.

The first major category of higher-order MRFs are *patch-based* priors, which model the statistics of small patches in an image. In Section 1.5.2 we discussed the Fields of Experts model of [77], which uses clique functions on each $k \times k$ patch of the image to create a generative model of image textures. This model consists of a set of k linear patch filters J_i , which are each run through a non-linear response function. The cost function is given by

$$f_C(\mathbf{x}_C) = \sum_{i=1}^k \alpha_i \log(1 + J_i^T \mathbf{x}_C). \quad (3.1)$$

We use machine learning approaches to find the best filters J_i and weights α_i , so that the resulting prior $p[\mathbf{x}]$ matches the observed distribution of image patches as closely as possible. As we observed in Section 1.5.2, using patch based priors like Fields of Experts for denoising leads to much less over-smoothing than the best available first-order models.

Due to (3.1) being continuously differentiable (treating the labels as continuous intensity values), Fields of Experts models are usually optimized using gradient descent or other continuous non-linear solvers. However, because of the large number of discrete labels for denoising problems (typically 256 discrete intensity levels), this model has been used as a benchmark in many comparisons of higher-order discrete solvers.

Other examples of patch-based priors include robust 3×3 Potts priors for im-

age denoising [55], 3×1 vertical patches used for geometric surface labeling [27], and size 3 learned patch priors for 3D object segmentation [98].

Related to patch-based priors like Fields of Experts, segment-based priors first segment the image into small, compact regions of similarly colored or textured pixels, and then add a prior that pixels within a segment should take similar labels. The Robust Potts (or P^n) model of [48] enforces consistency between the pixels in a segment by adding a cost for each differently labeled pixel within a segment, which is truncated to a fixed cost after enough differently labeled pixels occur. Cost functions on even larger groups of pixels than segments include the global costs such as the co-occurrence priors of [63], which improve semantic segmentation results by putting high costs on unlikely semantic categories both appearing in an image (for example, “cow” and “sofa”), while not penalizing common pairings (e.g., “cow” and “grass”).

The second major category of higher-order priors uses multiple variables to estimate higher-order derivatives of the image. We have already seen the curvature-based stereo model of [102], in Section 1.5.3. In order to remove the fronto-parallel bias of pairwise stereo priors, this model uses 3×1 cliques in the image, with a cost which uses a discrete approximation to the second-derivative of the depth map. By penalizing the second-derivative, this prior thus encourages the result to be locally planar. As we saw in Section 1.5.3, higher-order cliques are necessary to allow non-fronto-parallel planes in the depth image, as first-order priors can only penalize differences in neighboring depth values.

In general, natural models for penalizing the k th derivative require cliques of size $k + 1$. Other higher-order derivative priors include the second-derivative based model of [62] for non-rigid image registration, a 3rd derivative regularizer

for 1D signal processing [55], and many curvature-based models, including for binary segmentation [72], 2D and 3D surfaces [88] and segmentation and shape inpainting [85].

3.2 Inference Algorithms for Binary MRFs

Among computer vision inference methods, many of the most successful algorithms have been graph cuts methods. Large comparisons of inference algorithms across diverse datasets, including the Middlebury dataset [90], and OpenGM2 dataset [45], have found graph-cuts to be highly efficient for many problems in computer vision. In particular, the space of binary MRFs is particularly dominated by Graph Cut algorithms.

3.2.1 First-Order Submodular MRFs

For binary submodular problems (as we saw in Section 2.10), the max-flow/min-cut algorithm finds the exact global optimum in polynomial time. This efficient optimization was first used for vision problems by [11, 12], with several applications including image restoration and stereo problems. The connection between graph cuts and submodularity was explored in [52], where binary first-order polynomials with non-positive quadratic terms, $a_{i,j} \leq 0$, were termed *regular*.

A major reason for the empirical success of graph cuts are the specialized max-flow algorithms designed to have good performance on inputs typical of computer vision problems. For general graphs, flow algorithms based off

Push-Relabel[14] tend to have the best performance. However, computer vision graphs are generally constructed from priors which enforce locality between neighboring groups of pixels, and are typically on grids with edges only in a local neighborhood. Thus long augmenting paths tend to be rare. The Boykov-Kolmogorov flow algorithm [10] exploits this by maintaining two search trees in the residual graph, which grow towards each other from the source and sink. With short augmenting paths, these search trees can do incremental path finding from source to sink with minimal additional overhead. A small change to the Boykov-Kolmogorov algorithm, called Incremental Breadth-First Search [30], results in theoretically guaranteed runtime for the algorithm, as well as slightly improved (and currently state-of-the-art) empirical performance.

3.2.2 First-Order Nonsubmodular MRFs

For non-submodular first order MRFs, graph cuts no longer find the optimal solution. In fact, the problem is NP-complete (see [52] and Section 2.2.5). Despite the difficulty of finding optimal solutions, there are useful approximate inference algorithms. The most widespread of these is the Roof-Dual construction of [33], which maximizes a linear lower-bound to the discrete minimization problem. The Roof-Dual method was made practical for computer vision problems by [51], which used the graph construction of [8] in an algorithm called QPBO (for Quadratic Pseudo-Boolean Optimization). Like Graph Cuts for regular functions, QPBO uses the max-flow/min-cut algorithm. However, the underlying graph is twice as large.

The most important feature of QPBO is that it does not always return an

optimal labeling, but instead gives a *partial labeling* of the variables, meaning that some variables are labeled 0, 1, while others are unlabeled (which we will denote as “?”).

The Roof-Dual construction ensures that this partial labeling satisfies a property called *persistence* [33]. Persistence describes what happens when we fuse a partial labeling with another labeling. Let $\mathbf{x} \in \{0, 1, ?\}^V$ be a partial labeling, and $\mathbf{y} \in \{0, 1\}^V$ a complete labeling. We can stamp \mathbf{x} onto \mathbf{y} by replacing y_i with x_i for all the variables i where x_i is 0 or 1. That is, the fused labeling \mathbf{z} has $z_i = x_i$ whenever $x_i = 0, 1$ and $z_i = y_i$ otherwise. Then, a partial labeling $\mathbf{x} \in \{0, 1, ?\}^V$ is persistent if for any complete labeling \mathbf{y} , the fused result \mathbf{z} has weakly decreasing cost: $f(\mathbf{z}) \leq f(\mathbf{y})$.

Because the partial labeling from QPBO is persistent, we can derive several useful facts: First, there must be some optimal labeling \mathbf{y}^* which includes the labeled variables of \mathbf{x} . Indeed, let \mathbf{y}^* be any optimal labeling of f , and let \mathbf{z} be the fusion of \mathbf{y}^* with \mathbf{x} . Since $f(\mathbf{z}) \leq f(\mathbf{y}^*)$, we know that \mathbf{z} is also optimal, and it includes all the labeled variables from \mathbf{x} . Second, given our partial labeling, we can simplify our problem — since we know that there is an optimal solution containing the labeled variables from \mathbf{x} , we can now set these variables to be constant in f . This can eliminate unary and pairwise terms from f , and may separate the problem into disjoint connected components. More persistencies can be found recursively to get even more labeled variables, for example in QPBO with Probing [51] or Partial Optimality by Pruning [89].

Aside from QPBO, there are other approximate inference methods, which tend to be fast in practice, and which produce full (not partial) solutions, although without as many theoretical guarantees regarding the solution. A pop-

ular approach is to modify the energy function to remove the non-submodular edges. The simplest option, used first in [80] is to simply truncate positive quadratic terms to 0. This results in a submodular function which can be exactly optimized by graph-cuts, and if there are not too many submodular edges, then the optimal solution to the truncated energy may be close to optimal for the original function (however, there is no guarantee that it will be close).

A more sophisticated choice is to use an actual upper bound to the original function, as in the LSA-AUX method [31]. This method iteratively finds a series of upper bounds g^t , each of which has $g^t(\mathbf{x}^t) = f(\mathbf{x}^t)$ at the current point \mathbf{x}^t . Each move solves a pairwise submodular minimization problem $\mathbf{x}^{t+1} = \arg \min g^t(\mathbf{x})$ with graph cuts, giving a fast local-search method.

A final approach modifies the topology of the graph, instead of modifying the costs of each edge. If the graph is a tree (or, more generally, has a property called *low treewidth*), then we can find the exact optimal solution in linear time using dynamic programming. The method of [21] removes edges to get a low treewidth subgraph, while hopefully removing as few edges as possible.

3.2.3 Higher-Order Reductions

A popular strategy for using Graph Cuts with higher-order priors is to *reduce* the problem to a first-order one. A reduction to first order takes the original function $f(\mathbf{x})$, and via some transformations, including adding new variables y_i and edges, produces a function $g(\mathbf{x}, \mathbf{y})$ which satisfies $f(\mathbf{x}) = \min_{\mathbf{y}} g(\mathbf{x}, \mathbf{y})$. That is, if we minimize g over the auxiliary variables \mathbf{y} , then we get back our original function f . If we are trying to solve the minimization problem $\min_{\mathbf{x}} f(\mathbf{x})$, then

solving the joint minimization problem $\min_{\mathbf{x}, \mathbf{y}} g(\mathbf{x}, \mathbf{y})$ will give the same answer, and the same minimizer for \mathbf{x} .

For higher-order clique functions with particular forms, there are specialized reductions, which take advantage of structure in the cost function to make reducing higher-order to first-order relatively straightforward. These include concave cost functions [48], label consistency priors [49], cost functions which are sparse (only a small number of nonzero values) [78], and curvature regularization (including the stereo example of [102] discussed in Section 1.5.3).

For general higher-order functions, for a long time the only option was the method of reduction by substitution [76]. This method does not perform well in practice, as the resulting first-order MRF is particularly difficult to optimize. More recent reductions could handle multilinear polynomials with all negative coefficients [52, 24]. The first practical method for handling general higher-order functions is the Higher Order Clique Reduction (HOCR) algorithm of [37, 40]. This method produces a first order binary MRF for which QPBO can label large numbers of variables on typical vision inputs.

A different approach to finding higher-order reductions is Generalized Roof Duality (GRD) [43, 44], which proposed a class of submodular relaxations for an arbitrary higher-order MRF with clique size at most 4. GRD finds the best such relaxation by solving a linear program which searches over all possibilities from a characterization of quadratizable submodular functions from [104]. It then optimizes the relaxed function exactly using graph cuts. The relaxations found by GRD provide very good approximate solutions to the original higher-order problem. However, in addition to the restriction on the maximum clique size (which appears difficult to overcome) GRD is also computationally much

more intensive than HOCR.

Other general reduction strategies include the hypergraph-based reduction method of Chapter 4, and methods which choose between reduction strategies based on a separate inference problem [27].

3.2.4 Higher-order Submodular Functions

An alternative to reducing higher-order functions to first-order is to instead try to generalize flow algorithms to work directly on the higher-order functions.

We have already seen that the min-cut problem is only tractable when the associated cost function is submodular. The same holds true for higher-order functions. In fact, all submodular functions are exactly minimizable in polynomial time, with a current best asymptotic complexity of $O(n^6)$ [73]. Unfortunately, $O(n^6)$ is not practical for vision-sized inputs, so methods which are intermediate between min-cut and general submodular function minimization are necessary.

These methods are based off of Submodular Flow, which has been studied for some time in the combinatorial optimization literature [15, 26]. Submodular flow was adapted for higher-order binary MRFs by [53], which proposed an algorithm called Sum-of-Submodular (SoS) flow. This algorithm works for binary MRFs where each clique function is submodular (and hence the entire function is a “sum of submodular” functions). Recall that the sum of any number of submodular functions is still submodular, so general submodular minimization such as [73] could be used. However, by exploiting the clique structure of MRFs,

with cliques of size $k \ll n$, SoS flow is able to achieve similar runtime to existing standard flow algorithms.

Intuitively, the difference between max flow and sum-of-submodular flow is that in addition to capacity and conservation constraints, we also have constraints that the flow out of any set S is at most $g_C(S \cap C)$.

The first practical implementation of SoS function minimization was the Generic Cuts algorithm of [3], which used an augmenting paths algorithm for flow. This was followed by the SoS IBFS algorithm of [19] (described in Chapter 5) which generalized the currently state-of-the-art IBFS algorithm for vision problems to work for Sum of Submodular functions.

Finally, we also have analogues of the cost-modification approaches that we described above for non-submodular first-order problems. For higher-order functions, Auxiliary Cuts [6] uses convexity and other properties of image functionals to compute upper bounds which are pairwise submodular functions. These upper bounds are iteratively minimized and updated similarly to LSA-AUX [31].

The Pseudo-Bound method [92] extends this idea by considering a parameterized family of functions which include a pairwise submodular upper bound, and finding all minimizers of the entire family using parametric max-flow — by looking at all minimizers, a greater decrease in energy per iteration is obtained. Note that for both these methods, the upper bounds are all pairwise functions, even if the functions they approximate are higher-order.

A natively higher-order approach is to look for a Sum-of-Submodular upper bound to a non-submodular higher-order function. This approach will be

explored in Chapter 6.

3.3 Primal Algorithms

The next three sections will cover methods for optimizing multilabel MRFs. A useful categorization of these algorithms sorts them by how they utilize the Linear Programming relaxation (using the Local Marginal Polytope) and its dual. In this section, we'll cover algorithms that deal only with the original problem, and are therefore *primal* algorithms. More specifically, since they generally are only concerned with actual labelings and not the LP relaxation, they are primal-integral algorithms.

Most primal algorithms for optimizing MRFs have been some variant on local search. The basic recipe of these algorithms is to maintain some current state \mathbf{x}' , and then find the best solution \mathbf{x}' within some local neighborhood of \mathbf{x}' and update, $\mathbf{x}^{t+1} = \mathbf{x}'$.

The earliest algorithms for optimizing MRFs are essentially classic local search algorithms. Iterated Conditional Modes (ICM) [7] updates each variable x_i in turn, picking the best label given that all other labels remain constant, $x'_i = \operatorname{argmin}_{x_i} f(x_i, \mathbf{x}_{V-i})$. This is a simple local search with neighborhood size $|\mathcal{X}_i|$ at each iteration. However, it is prone to get stuck in poor local optima, since variables cannot vary simultaneously. More recent versions of ICM include Block ICM [46], in which blocks of variables are updated simultaneously. In grid-structured MRFs (when the edges of the MRF form a grid, such as the pixels of an image) dynamic programming can be used to efficiently optimize over very large blocks in linear time [13].

Simulated Annealing [28] searches in a small local neighborhood similar to ICM. However, instead of choosing the best label for x_i among all labels in \mathcal{X}_i , a random label is chosen, weighted towards choosing lower energy labels more frequently than higher energy labels. By sometimes choosing suboptimal labels, a larger space of solutions can be explored — in fact, by decreasing the *temperature* (the rate at which suboptimal solutions are chosen) slowly enough, simulated annealing will find the true global minimum, however convergence may take longer than a brute-force search of the entire label space \mathcal{X}^V .

For multilabel problems, graph-cuts methods are also local search algorithms, although over a much larger neighborhood. The most widely used graph cut techniques, including α -expansion [12] and its generalization, fusion moves [66], repeatedly solve a first-order binary MRF in order to minimize the original multilabel energy function. These move-making graph-cut algorithms maintain a current solution $\mathbf{x} \in \mathcal{X}^V$, and at each iteration propose a new solution $\mathbf{y} \in \mathcal{X}^V$. We get a binary problem by allowing each variable i to either keep its current label x_i , or switch to the new label y_i .

Both alpha-expansion and fusion moves are local search algorithms; however, the search neighborhood is much larger than usually encountered: because each variable has an independent binary choice, alpha-expansion can pick the best move among $|2^V|$ possible neighbors of \mathbf{x} .

More precisely, if we let S be the set of variables which switch labels, then the new, fused labeling is denoted $\mathbf{x}[S \leftarrow \mathbf{y}]$, which has label x_i for $i \in S$ and y_i for $i \notin S$. Then, from the original MRF energy f we get a binary function $g(S) = f(\mathbf{x}[S \leftarrow \mathbf{y}])$. This function g has the same clique structure as f , so we can optimize it using the techniques used for pseudoboolean functions described

above. Once we have found the optimal S^* for g (or some approximation of it, in case g is not submodular) we can update $\mathbf{x}^{t+1} = \mathbf{x}^t[S \leftarrow \mathbf{y}^t]$, and iterate.

Alpha-expansion [11, 12] is a specialization of the above (historically, the first to be considered) in which the proposal \mathbf{y} is a constant label, $y_i = \alpha$ for all $i \in V$. Alpha-expansion continues to be the most widely used graph-cuts algorithm for several reasons. It is simple to implement, as it doesn't require any special domain knowledge to pick the proposals \mathbf{y} . Furthermore, assuming that the clique functions $f_{i,j}$ are *metric*, meaning they satisfy the following properties

$$\begin{aligned}
f_{i,j}(a, a) &= 0 & \forall a \in \mathcal{X} \\
f_{i,j}(a, b) &> 0 & \forall a \neq b \\
f_{i,j}(a, b) &= f_{i,j}(b, a) \\
f_{i,j}(a, b) + f_{i,j}(b, c) &\leq f_{i,j}(a, c) \quad \forall a, b, c \in \mathcal{X}
\end{aligned} \tag{3.2}$$

then the binary subproblem g will always be submodular, and hence we can exactly find the optimal S^* using min-cut. Finally, α -expansion can have provable approximation bounds (in the sense of Definition 28, Section 2.2.5): when the $f_{i,j}$ are all Potts terms, with $f_{i,j}(x_i, x_j) = \lambda_{i,j}$ whenever $x_i \neq x_j$, and 0 otherwise, then alpha-expansion is always a 2-approximation [12]. When the $f_{i,j}$ are all the same, and form a metric, then the work of [47] showed that the approximation ratio is $2 \frac{f^{\max}}{f^{\min}}$ where f^{\max} is the maximum value of $f_{i,j}$ and f^{\min} is the minimum nonzero value of $f_{i,j}$.

For Fusion Moves [66], the binary subproblems may be non-submodular, in which case only approximate solutions to g can be found. However, proposals can be specifically chosen to better explore the label space \mathcal{X}^V . The original paper [66] proposed a number of variants, including Jump Moves, which search through label spaces for problems like Optical Flow, by allowing the current

label to move in horizontal or vertical translations from the current solution \mathbf{x} . The work of [38] proposed Gradient Descent Fusion Moves, for energy functions which are differentiable (in particular, for Fields of Experts) — the proposed move \mathbf{y} is along the direction of the energy gradient $\nabla f(\mathbf{x})$, which quickly moves \mathbf{x} towards lower energy solutions.

Finally, for special cases of energy functions, there are globally optimal solutions. In the case where the label set can be given an order $\mathcal{X}_i = \{a_1 < a_2 < \dots < a_\ell\}$, and the pairwise functions $f_{i,j}$ are all convex according to this order, then there is a graph construction due to Ishikawa [39] which finds the globally optimal labeling in a single min-cut solve. This condition has been generalized to a multi-label submodularity condition by [84], which similarly has a globally optimal solution.

3.4 Dual Algorithms

The class of dual algorithms all make use of the Local Marginal Polytope, and in particular, of the dual program described in Section 2.9. The Local Marginal Polytope was originally introduced by [86], and extended to the higher-order case by [99]. Recall that the dual program is a maximization over dual variables $\lambda_{C,i}(x_i)$, and that these dual variables can be interpreted as a reparameterization, in which some of the energy of the clique functions is partitioned among the unary terms.

Another way of interpreting the dual variables is as messages between variables. In particular, in the pairwise case, the dual variables $\lambda_{C,i}$ for an edge $C = \{i, j\}$ becomes a message from variable j to i , where $\lambda_{i,j}(x_i)$ communicates

some function of node j 's belief that the true state for i is x_i . For MRFs whose edges form a tree, this intuition can be made exact, with the Max-Product Belief Propagation algorithm [74]. For tree-structured MRFs, Max-Product BP can find the exact global optimum in linear time. Early message passing algorithms were developed without knowledge of the Local Marginal Polytope; however, global optimality can be proved by noting that Max-Product BP on a tree is actually a form of Dynamic Programming.

For MRFs which are not trees (for example, image grids) Max-Product BP can be applied to each variable in an iterative algorithm called Loopy BP [25]. When the MRF graph has loops, LBP may fail to converge, and has been observed to enter cycles, without even reaching a local optimum. However, it has achieved good performance in many practical problems, and was a widely-used alternative to graph-cuts methods.

The first provably convergent message passing algorithm is the Tree-Reweighted Sequential message passing algorithm of [50]. The analysis of TRW-S interprets the messages as reparameterizations of the energy function f , which provide a global lower bound. This lower bound is maximized, using max-product steps on subtrees of the graph. This lower bound is related to the Local Marginal Polytope, but using a different dual program (obtained by organizing the energy into higher-order terms on the subtrees before taking the dual).

Dual Decomposition [56] splits the objective into a set of overlapping terms, and uses subgradient ascent on the dual to enforce consistency among the labelings of the separate parts of the objective. In general, Dual Decomposition works for any splitting of the objective, so long as the subproblems can be exactly optimized efficiently. However, the subgradient ascent may take many

iterations to converge.

Other message passing algorithms with provable convergence explicitly use the Local Marginal Polytope dual, including Max-Sum Diffusion [100] and MPLP [29]. The latter two algorithms can be interpreted as block-coordinate ascent on the dual program. Because the dual is not smooth (it is piecewise linear) block coordinate ascent will converge to a solution, but it may not be dual-optimal. Methods to smooth the dual objective can converge to the optimal dual solution: these include Accelerated Dual Decomposition [42] and Adaptive Diminishing Smoothing [81].

All of the above methods are for first-order MRFs, however the same basic ideas translate to higher-order MRFs as well. A dual decomposition approach based on higher-order pattern-based priors, using Dynamic Programming as the optimizer for the subproblems, was proposed in [55]. Max-sum diffusion was applied to the higher-order case in [101], and TRW-S was similarly generalized to higher-order MRFs in [54]. These latter methods are based on the corresponding algorithms for first-order, using the higher order Local Marginal Polytope proposed in [99].

3.5 Primal-Dual Algorithms

Finally, Primal-Dual methods use both the primal and dual programs simultaneously. The connection between graph cuts and primal-dual techniques was established by [57] which showed that α -expansion could be interpreted as simultaneously optimizing primal and dual solutions.

The general recipe of primal-dual algorithms is that they iteratively update a primal integer solution (i.e., a labeling \mathbf{x}) similarly to alpha-expansion. However, they also maintain a dual solution λ , which guides the search during the binary min-cut solve. Furthermore, the primal and dual solutions are simultaneously updated, so as to satisfy invariants related to complementary slackness, which results in the final solution having provable approximation bounds. These technical details will be expanded on in Chapter 7.

The primal-dual algorithm of [57] overcomes the most important limitation of the α -expansion algorithm, which is the requirement that the pairwise energy must be a metric [12]. These methods also extend the approximation bounds for alpha-expansion with metric energies from [47]. The same approximation ratio still holds, but over a much broader class of energy functions.

Empirically, keeping track of the dual variables also allows a number of implementation speedups compared to α -expansion, resulting in the very efficient algorithm FastPD [59], which can be 3-9 times faster than alpha-expansion in practice.

For higher-order MRFs, the first primal-dual algorithm is the Sum-of-Submodular Primal Dual algorithm, SoSPD, which is covered in Chapter 7.

CHAPTER 4

HIGHER ORDER REDUCTIONS

We have already seen that first-order MRFs have well-understood and effective optimization algorithms, compared to higher-order MRFs. One way of bridging this gap is to find a way to transform higher-order MRFs into an equivalent first-order one. As described in Section 3.2.3, reduction methods can transform a binary MRF $f(\mathbf{x})$ to a quadratic function $g(\mathbf{x}, \mathbf{y})$, such that $f(\mathbf{x}) = \min_{\mathbf{y}} g(\mathbf{x}, \mathbf{y})$. Then, we can apply existing algorithms to the reduced first-order form g to get a solution to the higher-order original problem.

In this chapter, we will focus on binary MRFs, since the reduction methods discussed herein all work on binary problems only. Multi-label problems can be handled by repeated application of solving binary subproblems, as in (for example) alpha-expansion or fusion moves.

The main result of this chapter is a reduction method which exploits the hypergraph structure of the cliques C to transform a group of terms at once. For n binary variables, each of which appears in terms with k other variables, at worst we produce n non-submodular terms, while [37, 40] produces $O(nk)$. We identify a property (called local completeness) under which our method perform even better, and show that under certain assumptions several important vision problems (including common variants of fusion moves) have this property. We show experimentally that our method produces smaller weight of non-submodular edges, and that this metric is directly related to the effectiveness of QPBO [51]. Running on the same field of experts dataset used in [37, 40] we optimally label significantly more variables (96% versus 80%) and converge more rapidly to a lower energy. Preliminary experiments suggest that some other higher-order

MRFs used in stereo [102] and segmentation [1] are also locally complete and would thus benefit from our work.

4.1 Introduction

While graph-cuts are a popular method for solving first-order MRFs, such as the benchmarks described in [90] and [45], they are much more difficult to apply to higher-order MRFs. As a result, until recently this powerful optimization method has only been used for a few specialized higher-order MRFs, such as [48, 102].

The first general-purpose practical graph-cuts method for higher-order MRFs is that of Ishikawa [37, 40]. This method works by transforming the higher-order input MRF into an equivalent quadratic (pairwise) MRF by adding additional auxiliary variables and edges. The general class of such methods are known as *higher-order reductions* — this particular reduction is commonly referred to as Higher-Order Clique Reduction (HOCR). Since the resulting first-order MRF is non-submodular, it is optimized using QPBO, which produces a partial labeling (see Section 3.2.2). The quality of this partial labeling (i.e., the number of labeled pixels) is highly sensitive to the energy function.

A more theoretically-motivated approach to finding higher-order reductions is Generalized Roof Duality (GRD) [43, 44], which proposed a class of submodular relaxations for an arbitrary higher-order MRF with degree at most 4. GRD finds the best such relaxation by solving a linear program, and optimizes the relaxed function exactly using graph cuts. The relaxations found by GRD provide very good approximate solutions to the original higher-order problem. Beyond

the restriction on the MRFs degree, which appears difficult to overcome, GRD is also computationally much more intensive than HOCR.

In this paper we propose an alternative construction to HOCR and GRD, with improved theoretical and experimental performance. Instead of considering terms in the energy function one at a time, we make use of the fact that the clique structure of an MRF is a hypergraph, in order to reduce many terms at once. We will review existing reduction methods for solving higher-order MRFs with graph cuts in section 4.2, We present our new algorithm in section 4.3, and analyze its worst case performance in section 4.4. In section 4.5 we show that for problems with property called *local completeness* our method performs even better. Under certain assumptions we prove that some important vision problems are locally complete, including the fields of experts MRF considered by Ishikawa. Experimental results are given in section 4.7, along with experimental evidence that other vision problems [1, 102] are also locally complete.

4.2 Related work

There are a number of methods for reducing an arbitrary multilinear polynomial over binary variables into a quadratic one. The performance of the different methods is summarized in figure 4.1.

For all methods, we are interested in the size of the obtained quadratic function, including the number of additional vertices and edges required, as these directly affect the size of the min-cut problem which will be solved by QPBO. We make a particular note of the number of nonsubmodular edges as well as the weight of these edges, as these can negatively impact the solution returned

	New variables	Non-submodular edges	Submodular edges	Non-submodular weight
Substitution [76]	$O(nk)$	$O(nk)$	$O(nk)$	$O(nkM)$
Negative [24]	t	–	td	–
HOOR [37]	$O(td)$	$O(nk)$	$O(td^2)$	$O(d^2W)$
GRD [43] ($d \leq 4$)	$n + O(t)$	–	$O(td)$	–
Ours (worst case)	$n + O(td)$	n	$O(td^2)$	$O(dW)$
Ours (local completeness)	$n + O(t)$	n	$O(td)$	$O(dW)$

Figure 4.1: Resources required to reduce t terms of degrees up to d , for an energy function with n variables each of which occurs with up to k other variables. W is the total weight of all positive terms in the higher-order function. Unlike the other algorithms listed, GRD is only defined for terms of limited degree. There is no clear notion of non-submodular edges in the relaxation produced by GRD, so we mark these entries “–”. Non-submodular weight is the total weight of non-submodular edges in the reduced function.

by QPBO [90], as confirmed in our experiments¹.

4.2.1 Reduction by substitution

The original reduction method was introduced by Rosenberg [76]. The reduction operates on the multilinear polynomial representation of f . The algorithm iteratively eliminates all occurrences of some product $x_i x_j$ by introducing a new variable z , replacing $x_i x_j$ by z everywhere it occurs, and then adding the following penalty terms to the energy function: $Mx_i x_j - 2Mx_i z - 2Mx_j z + 3Mz$, where M is a suitably large constant. This forces z to take the value of $x_i x_j$ in any optimal solution.

¹Note that the total weight of nonsubmodular edges is not a perfect measure of the performance of QPBO. For example, many functions can have non-submodular edges, but after permuting some labels become submodular [84], and these functions can be exactly minimized by QPBO. Nevertheless, our experiments show this is a useful heuristic.

If each variable is in terms with at most k other variables, this reduction can be done with $O(nk)$ pairs, which results in $O(nk)$ new variables, $O(nk)$ non-submodular terms and $O(nk)$ submodular quadratic terms.

Note that the non-submodular terms have large coefficients. Experimentally it has been reported that QPBO performs very poorly on such energy functions (see, for example, [40, §8.3.4], which states that QPBO finds almost no persistencies).

4.2.2 Reducing negative-coefficient terms

Kolmogorov and Zabih [52] for $d = 3$ and Freedman and Drineas [24] for $d \geq 3$ suggested the following transformation for negative higher degree terms:

$$-x_1 \cdots x_d = \min_{y \in \mathbb{B}} y \left((d-1) - \sum_{j=1}^d x_j \right) \quad (4.1)$$

If we have t negative-coefficient terms of degree d , this gives t new variables and td submodular quadratic terms, but no non-submodular terms.

Let us note that the above equality remains valid even if we replace some of the x_j variables with their complements $\bar{x}_j = (1 - x_j)$. In [78] this was used to obtain a transformation for *sparse* functions, i.e., those with only a few labels \mathbf{x}_C have $f_C(\mathbf{x}_C) \neq 0$ (see type-II transformations in [78]).

4.2.3 Reducing positive-coefficient terms

The HOCR transformation [37, 40] was the first practical method for general higher-order functions. For a term of degree d , let $n_d = \lfloor \frac{d-1}{2} \rfloor$ and set $c_{i,d} = 1$ if

$d = i$ and i is odd, and $c_{i,d} = 2$ otherwise. Each positive term is reduced by

$$x_1 \cdots x_d = \min_{u_1, \dots, u_d} \sum_{i=1}^{n_d} u_i \left(c_{i,d} \left(- \sum_{j=1}^d x_j + 2i \right) - 1 \right) + \sum_{i < j} x_i x_j$$

For each term of degree d , we get $O(d)$ new variables. Each new variable is connected to each original variable by a submodular edge, for a total of $O(d^2)$ submodular edges. We also get non-submodular edges between all pairs of original variables x_i, x_j whenever x_i and x_j are in the same clique. If each variable occurs in terms with at most k other variables, then the number of non-submodular edges is $O(nk)$ (note that if the pair x_i, x_j occurs in multiple cliques, we only count the pair once, as they can be combined to a single edge in the flow network). Finally, if the positive term has positive weight $\alpha > 0$ then this term creates $\frac{d(d-1)}{2}$ non-submodular edges of weight α . So, if the total weight of positive terms is W , then the quadratic function has non-submodular edges of total weight $O(d^2 W)$.

Note that this reduction uses a large number of non-submodular edges: the d original variables are fully connected by positive weight edges. This is problematic, as it has been observed [90] that non-submodular edges can result in poor performance for graph cut optimizers like QPBO.

4.2.4 Generalized Roof Duality

Unlike the above methods, which are all rewrite rules for the individual terms of the multilinear polynomial, Generalized Roof Duality (GRD) [44] finds a reduction to quadratic form which is globally the best among a large class of candidates, called *submodular relaxations*.

GRD uses a characterization of all submodular functions expressible in quadratic form due to Zivny et. al. [104]. This reduction uses one additional variable for each term, as well as $O(d)$ additional edges for a degree d term.

The reduction also ensures the existence of *persistencies*, similar to the persistencies of Roof Duality [33] (and its implementation, QPBO [51]) whereby after solving the submodular relaxation, each variable is assigned a value in $\{0, 1, ?\}$, and every variable taking value 0 or 1 in the partial labeling actually takes that value in the (possibly unknown) global optimum.

To find the best submodular relaxation, GRD solves a linear program. Because GRD finds the tightest submodular relaxation, the returned labeling is typically of very high quality; however, solving the LP is computationally very expensive, making this algorithm impractical for large-sized problems. Instead of directly solving the LP, the authors also give heuristics to find nearly optimal submodular relaxations. These heuristic relaxations (denoted GRD-heur) also give very good labelings. However, we will show in our experiments that even these heuristic methods are several times slower than HOCR and our technique.

Finally, it is worth noting that GRD can only be applied when all terms have degree 3 or 4, and it is doubtful that the method can be generalized. The reduction [104] used by GRD to convert the submodular relaxation to quadratic form has only been described for functions of arity 4. Furthermore, writing down an LP for the optimal submodular relaxation requires being able to compactly describe the set of submodular functions with terms of degree d , and this task is NP hard for $d \geq 4$. In contrast, our method and Ishikawa’s have no restriction on the degree of terms involved.

4.3 Reducing groups of higher-order terms

The terms of a multilinear polynomial form a hypergraph \mathcal{H} . The vertices are the polynomial's variables, and there is a hyperedge $H = \{x_1, \dots, x_d\}$ with weight α_H whenever the polynomial has a term $\alpha_H x_1 \cdots x_d$. In contrast to earlier methods which reduce term-by-term, our new method uses this hypergraph structure to reduce a group of terms all at once.

The two theorems below are both concerned with reducing respectively all the positive or all the negative terms containing a single variable, (or small set of variables); we will write this common subset of variables as U . The most important special case of our reduction is shown in figure 4.2, where we consider all positive terms which contain the variable x_1 , i.e., $U = \{x_1\}$.

Theorem 58. *Let \mathcal{H} be a set of terms such that each $H \in \mathcal{H}$ contains U , i.e., $U \subseteq H$. Furthermore, we require all the hyperedges H have positive weights $\alpha_H > 0$. Let $f(x) = \sum_{H \in \mathcal{H}} \alpha_H \prod_{j \in H} x_j$ be this polynomial. Then $f(x)$ is equal to*

$$\min_{y \in [0,1]} \left(\sum_{H \in \mathcal{H}} \alpha_H \right) y \prod_{j \in U} x_j + \sum_{H \in \mathcal{H}} \alpha_H \bar{y} \prod_{j \in H \setminus U} x_j. \quad (4.2)$$

Proof. Given any assignment of the variables x_1, \dots, x_n , either (1) all the variables in U are 1, or (2) some variable in U is 0.

Case 1: Substituting 1 for the variables in U , $f(x)$ is equal to $\sum_{H \in \mathcal{H}} \alpha_H \prod_{j \in H \setminus U} x_j$ and (4.2) is $\min_y (\sum_{H \in \mathcal{H}} \alpha_H) y + \sum_{H \in \mathcal{H}} \alpha_H \bar{y} \prod_{j \in H \setminus U} x_j$. If we assign $y = 1$, then (4.2) becomes $\sum_{H \in \mathcal{H}} \alpha_H$, and if we assign $y = 0$, then it becomes $\sum_{H \in \mathcal{H}} \alpha_H \prod_{j \in H \setminus U} x_j$. This quantity is always less than or equal to $\sum_{H \in \mathcal{H}} \alpha_H$, so the minimum is achieved when $y = 0$, in which case, $f(x)$ equals (4.2).

Case 2: The product $\prod_{j \in U} x_j$ is 0. Since all the terms of $f(x)$ share the common subset U , $f(x) = 0$. Similarly, (4.2) is $\sum_{H \in \mathcal{H}} \alpha_H \bar{y} \prod_{j \in H \setminus U} x_j$. If we assign $y = 1$, then this sum is 0, whereas if we assign $y = 0$, then it is positive, since each α_H is positive. Thus, the minimum is achieved when $y = 1$, in which case (4.2) is 0 hence equal to $f(x)$.

□

For every positive term containing the common subset U , equation (4.2) replaces it with a new term $\alpha_H \bar{y} \prod_{j \in H \setminus U} x_j$. To get a multilinear polynomial, we replace the negated variable \bar{y} with $1 - y$, which splits each term into two:

$$\alpha_H \prod_{j \in H} x_j = \alpha_H \prod_{j \in H \setminus U} x_j - \alpha_H y \prod_{j \in H \setminus U} x_j \quad (4.3)$$

Corollary 59. *When we apply equation (4.2) to a positive term, we obtain a positive term of smaller degree, and a negative term with y replacing the common subset U .*

For reducing the negative-coefficient terms all sharing some common subset, we have a similar theorem.

Theorem 60. *Consider \mathcal{H} and U as above, where now the coefficients α_H are negative for all H . Let g be the corresponding polynomial. Then for any assignment of the variables, $g(x)$ is*

$$\min_{y \in \{0,1\}} \sum_{H \in \mathcal{H}} -\alpha_H \left(1 - \prod_{j \in U} x_j - \prod_{j \in H \setminus U} x_j \right) y \quad (4.4)$$

Proof. The proof is similar to the proof for Theorem 58. The minimum is achieved when $y = \prod_{j \in U} x_j$. □

A crucial difference between this reduction and theorem 58 is that in the positive case, we could let the common subset U be a single variable. However,

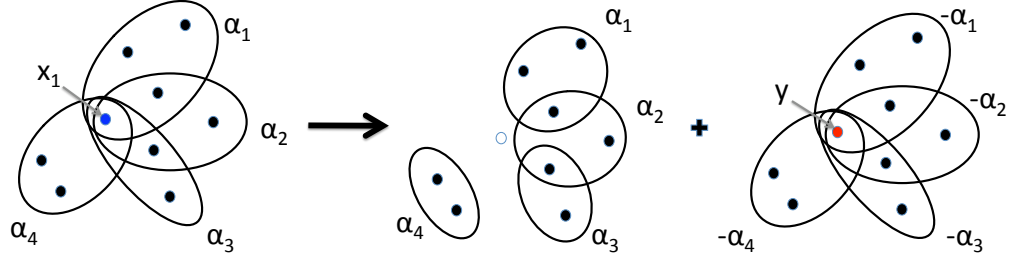


Figure 4.2: Our main reduction. At left are all the original positive terms containing the common variable x_1 (so $\alpha_i > 0$). At right are all the new terms we obtain from equation (4.2). The positive terms on top are just the original terms minus x_1 , and the negative terms on bottom are the original terms with y replacing x_1 .

applying Theorem 60 to $U = \{x_i\}$ removes the term $\alpha_H \prod_{j \in H} x_j$ and replaces it with $\alpha_H y \prod_{j \in H \setminus \{1\}} x_j$, another negative term of the same degree. Trying to apply this reduction repeatedly will thus never terminate. However, if U consists of two or more variables, then grouping all terms containing U and reducing results in smaller degree terms replacing every term that we start with.

4.3.1 Our method

Equations (4.2) and (4.4) can be used for different reduction strategies. Both depend upon the choice of common variables U . Besides choosing $|U|$, we can also decide the order to consider different choices of U ; for example, which single variable to use to apply equation (4.2), or which pair of variables to use to apply equation (4.4).

We will focus on the simplest case: we let the common part U be a single variable x_i , and reduce positive terms containing this variable via equation (4.2).

Negative terms will be reduced using the method of section 4.2.2. Note that more complicated schemes are also possible, such as picking pairs of variables and reducing both positive and negative terms containing this pair via equations (4.2) and (4.4).

Our method reduces a multilinear polynomial with higher-order terms, to quadratic form in two steps:

Step 1. Eliminate all higher-order positive terms by repeated application of Theorem 58, with the common subset U set to a single variable x_1 . Gather all terms containing x_1 , and replace them with equation (4.2). If \mathcal{H} consists of all positive terms containing x_1 , then

$$\sum_{H \in \mathcal{H}} \alpha_H \prod_{j \in H} x_j = \min_{y \in \{0,1\}} \left(\sum_{H \in \mathcal{H}} \alpha_H \right) x_1 y \quad (4.5a)$$

$$+ \sum_{H \in \mathcal{H}} \alpha_H \prod_{j \in H \setminus \{1\}} x_j \quad (4.5b)$$

$$- \sum_{H \in \mathcal{H}} \alpha_H y \prod_{j \in H \setminus \{1\}} x_j \quad (4.5c)$$

The positive terms now form a hypergraph on one fewer variable, so repeat with x_2, \dots, x_n until all positive terms are reduced.

Step 2. All higher-order terms now have negative coefficients. Reduce them term-by-term using the methods in section 4.2.2.

Note that equation (4.5) is simply the special case of equation (4.2) for a single variable. This special case is illustrated in figure 4.2.

4.4 Worst case performance

The results of applying equation (4.5) consist of three parts: a positive quadratic term (4.5a); and for each term, a positive term on the original variables with x_1 removed (4.5b); and a negative term with y replacing x_1 (4.5c).

Note that in the course of the reduction, we may create a monomial on some variables x_1, \dots, x_d and another monomial on the same variables already existed in the input. In this case, we get lucky, since we can just sum the new term's coefficient with the existing monomial, which doesn't increase the size of the representation. To analyze the worst-case performance, we will assume that this never happens. In section 4.5 we will revisit this possibility.

Under this assumption, each positive term of degree d that we start with will have a single variable removed every time we apply the reduction. To be fully reduced it must go through $d - 1$ applications of the rule, producing negative terms of degrees $2, \dots, d$. Reducing these $d - 1$ negative terms by section 4.2.2 results in $O(d)$ new variables and $O(d^2)$ submodular quadratic terms.

Overall, to reduce t positive terms of degree d on n variables, in the worst case our method requires $n + O(td)$ new variables, $O(td^2)$ submodular terms and at most n non-submodular terms. Even in the worst case our algorithm's asymptotic performance is similar to HOCR (see figure 4.1). However, our method produces at most n non-submodular terms, compared to $O(nk)$ for HOCR.

For the weight of non-submodular edges, each positive term contributes α_H to (4.5a) each time it is reduced, for a total of $(d-1)\alpha_H$ weight in non-submodular edges. If the total weight of positive terms is W , we get $O(dW)$ non-submodular

weight in the reduced form, a factor d improvement over HO CR.

4.5 Local completeness

We can improve on this worst-case analysis for some common vision problems such as [77, 102]. We have identified a property of certain energy functions that we call local completeness, where our algorithm (unlike HO CR or GRD) has improved asymptotic performance. The basic idea behind local completeness is that whenever a monomial $x_1 \cdots x_d$ occurs in the input, we are also likely to see all the monomials on all subsets of these d variables as well. In essence, local completeness argues that typical inputs to vision problems are “bad” in the sense of having lots of terms. If a certain problem is locally complete, then this is a lower-bound argument to show that the input is necessarily large, and hence methods (such as ours) which exploit the shared structure of the graph will be more successful.

To be precise, consider a multilinear polynomial on the binary variables x_1, \dots, x_n , and denote by \mathcal{H} the hypergraph of its monomials, as before. Note that \mathcal{H} is not necessarily in minimal form (i.e., if $H \subseteq H'$ then both H and H' may both be hyperedges in \mathcal{H}). Let \mathcal{H}' be the “completed” hypergraph, formed by all subsets of edges in \mathcal{H} (that is, $\mathcal{H}' = \bigcup_{H \in \mathcal{H}} 2^H$).

Definition 61. *A polynomial is locally complete with completeness c (or has local completeness c) if $|\mathcal{H}| \geq c|\mathcal{H}'|$ for some $c \in (0, 1]$.*

To explain the terminology, note that the larger hypergraph \mathcal{H}' is obtained by *completing* our input \mathcal{H} , to include all the subsets of every term that we started

with.

Every polynomial is locally complete for some completeness c , as we can always choose $c = \frac{|\mathcal{H}|}{|\mathcal{H}'|}$. However, we are interested in classes of problems which remain complete as the problem size grows, so we say that *a family of polynomials is locally complete* if there is a fixed c such that all the polynomials have local completeness c . For example, a family P of polynomials arising from a particular vision problem would be locally complete if we always had $1/2$ of all subsets of terms appearing in all instances of P .

4.5.1 Performance on locally complete problems

Recall the procedure for reducing positive terms, using equation 4.5. We would like the extra positive term we create, with variables $H \setminus \{1\}$, to combine with some existing term. If it happens that $H \setminus \{1\}$ is already a term with coefficient $\beta_{H \setminus \{1\}}$, then we add α_H to this coefficient, and do not create a new term.

This motivates the definition of local completeness: the new positive terms in (4.5b) have variables which are subsets of our original terms, so if our energy function has local completeness c , the new positive terms will combine with existing terms a fraction c of the time.

Theorem 62. *If an energy function has local completeness c , the procedure of Section 4.3.1 for reducing positive terms will result in at most $\frac{1}{c}|\mathcal{H}|$ negative coefficient terms*

Proof. By the definition of local completeness $|\mathcal{H}'| \leq \frac{1}{c}|\mathcal{H}|$. As a notational convenience, add in all the extra subsets contained in \mathcal{H}' as monomials with co-

efficient 0, so there are now $|\mathcal{H}'|$ terms. Having done this, since \mathcal{H}' is closed under subsets, the positive terms produced by (4.5b) will always combine with existing terms.

Applying equation 4.5 removes the term $\alpha_H \prod_{j \in H} x_j$, changes the coefficient on the term with variables $H \setminus \{1\}$, and adds a new negative term $\alpha_{HY} \prod_{j \in H \setminus \{1\}} x_j$. The total number of terms remains constant.

Therefore, when we have finished reducing all positive terms, we are left with only negative terms, and we have as many as we started with, namely $|\mathcal{H}'| \leq \frac{1}{c}|\mathcal{H}|$. \square

If we started with t terms of up to degree d on n variables, the entire reduction results in at most $n + \frac{1}{c}t$ new variables, $\frac{1}{c}td$ submodular terms and n non-submodular terms. For a family of locally complete inputs, c is constant, giving the asymptotic results in figure 4.1.

Local completeness is stronger than strictly necessary — we only really need that when reducing terms containing x_1 , all the terms $H \setminus \{1\}$ already exist. In this case, the analysis depends on the order in which we pick variables. Local completeness gives the stronger property that no matter what order we choose variables to reduce, we always get a large fraction of terms combining.

Finally, we reiterate that local completeness does not state that such problems are easy to solve, but rather the opposite: such problems (which include many vision problems, as shown below) have intrinsically large representations as multilinear polynomials; but nevertheless, in such cases our reduction uses few additional variables and edges.

4.6 Locally complete energy functions in vision

We can show that under some reasonable assumptions an important class of vision problems will have locally complete energy functions. Specifically, we consider fusion moves [66] under an FoE prior [77] with random proposals, as used as a benchmark for HOCR in [37, 40].

The original (non-binary) energy function can be written as a sum over cliques C in the image $\sum_C f_C(\mathbf{x}_C)$. A single fusion move has an input image I and a proposed image I' , and for every pixel there is binary variable that encodes whether that pixel takes its intensity from I or I' . This results in a binary energy function on these variables to compute the optimal fusion move.

We can better analyze fusion moves by moving to a continuous framework. Embed the original intensities in \mathbb{R} , and extend the clique energies f_C to functions on \mathbb{R}^d . We need two assumptions: (1) f_C is $d - 1$ times continuously differentiable and (2) each of the d different mixed partials $\frac{\partial^{d-1} f}{\partial x_1 \dots \widehat{\partial x_i} \dots \partial x_d}$ (where $\widehat{\partial x_i}$ means to omit the i -th partial) take their zeros in a set of measure 0.

Theorem 63. *Under these two assumptions, the set of proposed-current image pairs (I, I') for which the fusion move binary energy function does not have local completeness 1 has measure 0 as a subset of $\mathbb{R}^n \times \mathbb{R}^n$.*

We defer the proof of this theorem to Appendix A, and provide a proof sketch. We write the fusion move binary energy function in terms of n binary variables b_i . Writing this as a multilinear polynomial in b , each clique C can result in terms t_S for each subset S of C . We can show that the energy function is locally complete, if the coefficient on t_S is almost never (i.e., with probability 0) zero.

For example, here is how to calculate the coefficient on the term b_1b_2 in a clique of size 3. If I_1, I_2, I_3 are the labellings in the current image on C , and I'_1, I'_2, I'_3 are the proposed labellings, then the coefficient on b_1b_2 is

$$f_C(I_0, I_1, I_2) - f_C(I'_0, I_1, I_2) - f_C(I_0, I'_1, I_2) + f_C(I'_0, I'_1, I_2) \quad (4.6)$$

Since the labels are in \mathbb{R} , the four 3-pixel images mentioned in this coefficient lie on a rectangle in \mathbb{R}^3 . If we give each of these points v heights of $f_C(v)$, then 4.6 is 0 if and only if the four points are coplanar.

In general, we do not expect 4 arbitrary points to lie on a plane. However, if f_C has no curvature, then any 4 such points will be coplanar. In the full proof, we show that if there exists any open ball of image pairs with zero coefficient on b_1b_2 , then the energy function is flat ($\frac{\partial^2 f}{\partial x_1 \partial x_2} = 0$) in the same ball (contradicting our assumption that the partials are nonzero almost everywhere). We also extend this to larger degree terms, to prove the general case.

Corollary 64. *The energy functions obtained from fusion moves with FoE priors and proposals chosen as random images are locally complete with probability 1.*

Proof. The functions f_C for the FoE model given in [77] are infinitely differentiable, and their mixed partials have their zeros in a set of measure 0. Since the proposed images are chosen from a continuous distribution over \mathbb{R}^n , events of measure 0 occur with probability 0. \square

4.7 Experimental results

We have provided a freely available open source implementation of our algorithm at <http://www.cs.cornell.edu/~afix/software.html>. We ex-

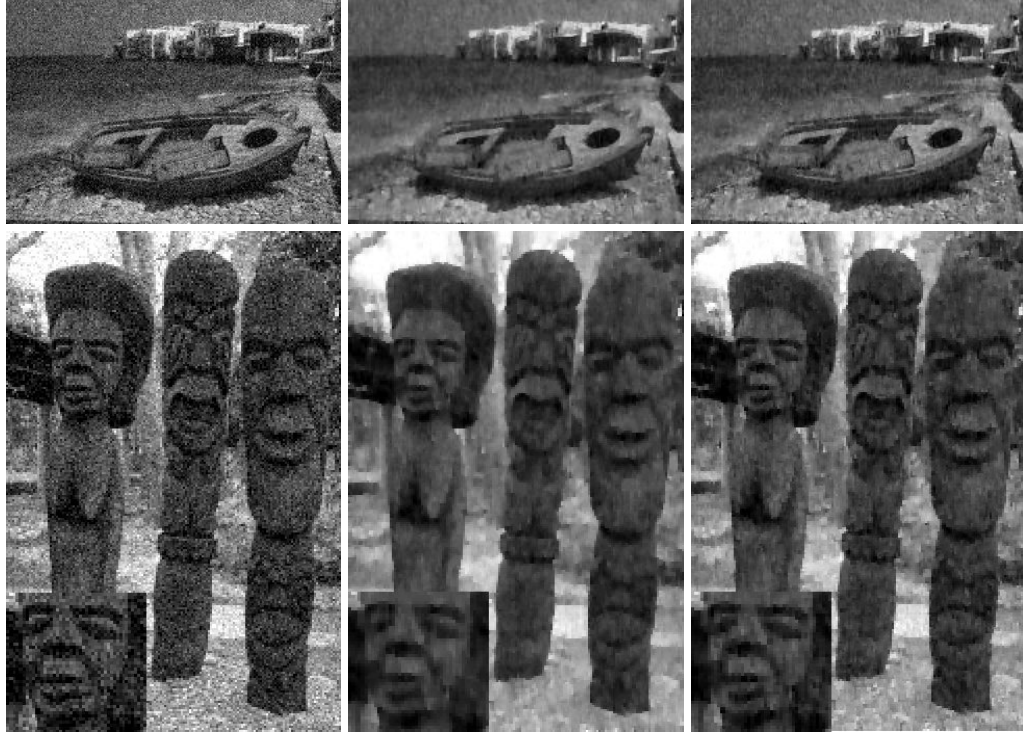


Figure 4.3: Denoising examples. At left is the noisy input image, with our result in the middle and Ishikawa's at right. Results are shown after 30 iterations. More images are included in the supplemental material. To compare energy values with visual results, the images on the top row have energies 118,014, 26,103 and 38,304 respectively; those on the bottom have energies 118,391, 25,865 and 38,336.

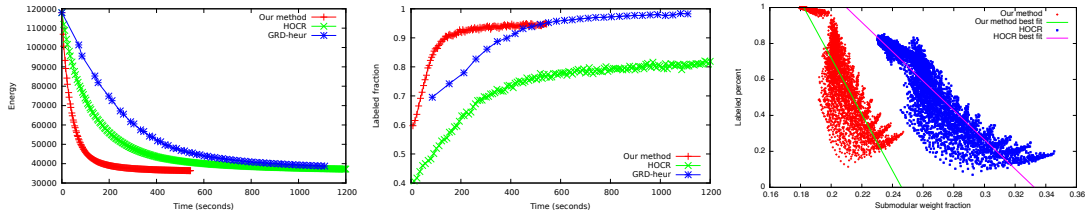


Figure 4.4: Energy after each fusion move (left), and percentage of pixels labeled by QPBO (center), for the image at top of figure 4.3. Other images from [40] give very similar curves. (right) Fraction of pixels labeled by QPBO vs total weight of non-submodular edges (as a fraction of the total weight of all edges), along with best-fit lines for each method.

perimentally compared our reduction with the two available, general-purpose higher-order reductions: HOCR [37, 40] and GRD [43, 44]. These three methods are all direct competitors in the types of energy functions they can handle (with the limitation that GRD is restricted to $d \leq 4$). All methods have publicly available code implemented in C++, and provide very similar interfaces for setting up and optimizing a higher-order MRF.

For all experiments, we only report the results from the heuristic version of GRD, GRD-heur. Because the exact version solves an LP, running this method on vision-sized inputs proved to be prohibitive. A single iteration of fusion move took an average of an hour, compared to 40 seconds for GRD-heur. Consequently, it was impossible to run this method on the full dataset. Fortunately, GRD-heur has been shown [43, 44] to have similar optimization quality to the exact GRD, at the gain of significantly less computation time.

Our benchmark for evaluating the methods is the Fields of Experts prior for image denoising with fusion moves, using a dataset of 200 images. This same experiment was used in the original evaluation of HOCR [37], and later in the evaluation of GRD [43]. We use the same MRF, and as similar energy functions as possible. The fusion moves alternate between a randomly generated uniform image and a blurred image, and the energy function has clique size 4. We do multiple fusion moves on multiple images, so the effects of randomness are minimal.

To compare the effectiveness of each method on the individual binary sub-problems, we ran all three algorithms on the first 30 fusion moves for each image, using the same starting point and proposal for each method (we averaged over only 30 iterations, because later iterations reduce the energy by much

smaller amounts, and wash-out the differences in the methods). The results, averaged over the $200 \times 30 = 6000$ fusion moves are summarized in figure 4.6. Overall, we see that our method is strictly preferable to HOCT in all metrics, giving a better energy improvement per fusion move, labeling more pixels in QPBO, and taking less time. Compared to GRD, we see that GRD does label more pixels, and reduces the energy by an additional 35%; however, it takes over 7 times as long per iteration to compute.

To compare the effect of total-weight of non-submodular edges, we plotted the fraction of pixels labeled by QPBO vs. the fraction of non-submodular weight (divided by the total weight of all edges) in figure 4.4. There is a clear negative relation between these quantities — best fit lines had slopes of -15.6 and -8.2 for our reduction and HOCT respectively. Correspondingly, our method had a lower average fraction of non-submodular weight, 19.7% vs 26.0%, and a higher fraction labeled by QPBO, 76.1% vs 59.4%.

We also tested the effect of choosing which order to reduce variables. The default for all experiments was to reduce the variables in order x_1, \dots, x_n . An alternative would be to choose the variables in decreasing order of number of positive terms. As predicted by our theoretical analysis, this difference had no measurable benefit. The “smart ordering” had 1% more average energy reduction, 0.4% fewer variables labeled, and won against the standard ordering (in terms of energy reduction) in only 42% of the 6000 fusion-moves. We also tested a random ordering, which performed somewhat worse than the standard ordering (better in only 2% of fusion moves, and 2% worse energy reduction). Because of extra bookkeeping, the smart ordering took 51% longer on average — consequently, we recommend using the standard order over more complicated

	Final energy	Time (seconds)
HOCR	32,199 (+2.3%)	2,050 (+102%)
GRD-heur	31,375 (−0.3%)	5,587 (+450%)
Our method	31,473	1,012

Figure 4.5: Comparison of end-to-end performance on benchmarks in [40] at convergence of the fusion move optimization, averaged over all images. Relative performance compared to our method is shown in parenthesis.

	Energy improvement	Percent labeled by QPBO	Time (seconds)
HOCR	1,302 (−45%)	59.4% (−22%)	14.1 (+150%)
GRD-heur	3,183 (+35%)	86.3% (+13%)	40.2 (+620%)
Our method	2,351	76.1%	5.6

Figure 4.6: Performance comparison of reductions, on benchmarks in [40], averaged over 30 iterations of fusion move. Relative performance compared to our method in parenthesis.

schemes.

As a second experiment, we compared the end-to-end performance of using each optimizer all the way through a complete run of the fusion-move algorithm. These results are displayed in figures 4.4 and 4.5.² Our method converges much faster, despite GRD reducing the energy by slightly more each step. Overall, the computational inefficiency of GRD greatly outweighs the marginal

²We averaged together pairs of consecutive fusion moves in the graph shown at right in figure 4.4. This avoids the distracting sawtooth pattern visible in [37, 40], due to the alternation between random fusion moves and blurred fusion moves.

	Extra variables	Non-submodular terms	Total terms
HOCR	224,346	421,897	1,133,811
Our method	236,806 (+6%)	38,343 (−90%)	677,183 (−40%)

Figure 4.7: Total size of reductions, on Ishikawa’s benchmarks in [40]. Relative performance of our method in parenthesis.

improvement in per-subproblem solution quality for fusion move.

The sizes of the obtained reductions for our method and the other term-rewriting reduction, HOCR, are summarized in figure 4.7. Overall, our method does better in practice than the asymptotic analysis in figure 4.1 suggests. As predicted, we produce many fewer non-submodular terms, but we also produce fewer submodular terms (a relative improvement of 10%).

Visual results are shown in figure 4.3. In the boat image our results appear more accurate in smooth areas like the water, and the face image (shown magnified at bottom left) is also noticeably smoother. These results are after 30 fusion moves. The images after convergence (shown, along with more examples, in the supplemental material) are visually similar, though we still obtain lower energy.

Finally, we experimentally computed the local completeness of two early vision problems that are quite far from denoising, namely stereo [102] (clique size 3) and segmentation [1] (clique size 4). We analyzed the binary energy functions produced from 60 iterations of [102]. These energy functions have a very high local completeness; on average the energy functions are c -complete for $c = .98$, and their least locally complete energy function had $c = .96$. We also discovered that the higher-order segmentation energy function of [1] is absolutely locally complete ($c = 1$). These results suggest that our method may be particularly well suited to a number of important vision problems.

CHAPTER 5

SUM OF SUBMODULAR MINIMIZATION

The reduction methods of the previous chapter provide an algebraic method for transforming higher-order binary MRFs to first-order. However, while this transformation preserves some features of the original energy function (e.g., the global minimum value remains the same) the resulting first-order functions are not always easily solvable. In particular, we have seen that non-submodular terms in the resulting reduced energy can lead to poor solutions from optimizers like QPBO.

An alternate approach to minimizing binary MRFs is to apply flow algorithms directly to the higher-order energy. Our goal is to preserve two key properties of max-flow based solvers: (1) global optimality of the solution obtained and (2) fast performance on typical inputs for vision problems. To do this, we will need a generalization of max-flow to the higher-order case — this generalization is called Sum-of-Submodular flow.

Sum-of-Submodular flow, and the corresponding cut problem, Sum-of-Submodular minimization, occupy a middle ground between the max-flow min-cut problem, and general submodular function minimization. A set function $f : 2^V \rightarrow \mathbb{R}$ is Sum-of-Submodular (SoS) if it is a sum

$$f(S) = \sum_{i \in S} f_i + \sum_{i \notin S} f'_i + \sum_C f_C(S \cap C) \quad (5.1)$$

where each clique function f_C is submodular.

Recall that a sum of submodular functions is itself submodular, so this is a special case of general submodular function minimization. However, we will

show that the structure of f (in particular, that the cliques C form a hypergraph) allows much faster minimization than the $O(n^6)$ algorithm of [73]. Additionally, we also have that standard min-cut is a special case of SoS minimization, since for each directed arc (i, j) with capacity $c_{i,j}$, the cost of that edge being cut can be written as a submodular clique function

$$f_{i,j}(S) = \begin{cases} c_{i,j} & i \in S, j \notin S \\ 0 & \text{otherwise} \end{cases} \quad (5.2)$$

So, SoS minimization lies in between min-cut and general submodular minimization:

$$\text{MIN-CUT} \subseteq \text{SOS MINIMIZATION} \subseteq \text{SUBMODULAR MINIMIZATION} \quad (5.3)$$

In this chapter, we describe how a Sum-of-Submodular (SoS) function can be minimized by means of an SoS flow network, and give a fast algorithm for solving this minimization, as originally presented in [19].

Throughout this chapter, we will assume that f is a set function, which is a sum of unary and clique functions of the form (5.1). We will also assume that f has been reparameterized such that $f_C \geq 0$ and $f_C(\emptyset) = f_C(C) = 0$, and the linear terms have $f_i, f'_i \geq 0$.

5.1 Sum of Submodular Minimization via Submodular Flow

For SoS functions, the cut problem is easy to describe: minimize $f(S)$ over all sets $S \subseteq V$. In this section, we detail the dual problem: Sum-of-Submodular flow.

5.1.1 Definitions and Graph Construction

Submodular flow has existed in the combinatorial optimization literature for some time [15, 26]. However, these algorithms are designed for full-order (or n th order) submodular functions, meaning there is no internal clique structure on the function f . The work of [53] was the first to develop an algorithm for the clique structured case (SoS flow), and the problem formulation and mathematical notation of this section are based on that work.

SoS flow is similar to the max-flow problem, in that there is a network of nodes and arcs on which we want to push flow from s to t . However, the notion of residual capacity will be slightly modified from that of standard max-flow.

We begin with a network $G = (V \cup \{s, t\}, A)$. We will denote $V + s + t$ by \bar{V} . As in the max-flow reduction for Graph Cuts, there are source and sink arcs (s, i) and (i, t) for every $i \in V$. Additionally, for each clique C , there is an arc $(i, j)_C$ for every pair $\{i, j\} \in C$.¹

Every arc $a \in A$ also has an associated residual capacity \bar{c}_a . The residual capacity of arcs (s, i) and (i, t) are the familiar residual capacities from max-flow: these arcs have starting capacities $c_{s,i}$ and $c_{i,t}$ (determined by the unary terms of f), and whenever we push flow on a source or sink arc, we decrease the residual capacity by the same amount.

For the interior arcs, we need one further piece of information. In addition to residual capacities, we also keep track of residual clique functions $\bar{f}_C(S)$, related to the flow values by the following rule: whenever we push δ units of flow on

¹To explain the notation, note that $\{i, j\}$ might be in multiple cliques C , so we may have multiple edges (i, j) (that is, G is a multigraph). We distinguish between them by the subscript C .

arc $(i, j)_C$, we update $\bar{f}_C(S)$ by

$$\bar{f}_C(S) \leftarrow \begin{cases} \bar{f}_C(S) - \delta & i \in S, j \notin S \\ \bar{f}_C(S) + \delta & i \notin S, j \in S \\ \bar{f}_C(S) & \text{otherwise} \end{cases} \quad (5.4)$$

A flow ϕ is a function $\phi : A \rightarrow \mathbb{R}^{\geq 0}$ which satisfies the usual conservation constraints (meaning flow into a node i is equal to the flow out of i for all $i \neq s, t$). The residual clique functions \bar{f}_C result from applying (5.4) each time an arc has δ units of flow, so we have that

$$\bar{f}_C(S) = f_C(S) - \sum_{i \in S, j \notin S} \phi_{i,j,C} + \sum_{j \in S, i \notin S} \phi_{i,j,C} \quad (5.5)$$

That is, the residual clique function $\bar{f}_C(S)$ is the original clique function $f_C(S)$, minus the outflow from S along arcs in C , plus the flow into S along arcs in C .

The residual capacities of the interior arcs are chosen so that the \bar{f}_C are always nonnegative. Accordingly, we define $\bar{c}_{i,j,C} = \min_S \{\bar{f}_C(S) \mid i \in S, j \notin S\}$. A flow is feasible if all residual capacities are nonnegative.

5.1.2 Flow as a Reparameterization

The key to understanding Sum-of-Submodular flow is that any flow gives a reparameterization of the original cost function. That is, flows in the graph defined above are just different ways of re-writing the function f . Finding the maximum flow will end up with a reparameterized f which is particularly easy to optimize, leading to an algorithm for finding the global minimum of f .

Interestingly, this same idea is used in several algorithms for energy minimization, including QPBO (see [51] for a very accessible review of this idea) and

the dual-LP methods for optimizing multilabel MRFs described in Section 3.4. We will also see this idea in the development of the SoSPD algorithm of Chapter 7.

To define this reparameterization, we first define the residual cost function $\bar{f}(S)$ to be

$$\bar{f}(S) = \sum_{i \in S} \bar{c}_{i,t} + \sum_{i \notin S} \bar{c}_{s,i} + \sum_C \bar{f}_C(S \cap C) \quad (5.6)$$

and also define the value of the flow ϕ to be the total outflow of the source:

$$v(\phi) = \sum_{i \in V} \phi_{s,i} \quad (5.7)$$

Lemma 65. *Any flow ϕ gives a reparameterization of the original cost function, with*

$$f(S) = v(\phi) + \bar{f}(S) \quad (5.8)$$

for all feasible flows ϕ and all sets $S \subseteq V$.

Recall that we defined a flow ϕ to be feasible whenever $\bar{f}_C \geq 0$ and $\bar{f}_i, \bar{f}'_i \geq 0$ for all C and i . In particular, if ϕ is feasible then $\bar{f}(S) \geq 0$ for all S , which immediately gives the following lower bound on the minimum of f .

Corollary 66. *If ϕ is feasible, then*

$$f(S) \geq v(\phi) \quad (5.9)$$

for all $S \subseteq V$.

Proof of Lemma 65. As with most proofs of one function being a reparameterization of another, we have that various quantities have been added and subtracted to the energy in a way that cancels out. In this case, we expand out the residual

capacities in (5.6):

$$\begin{aligned}
\bar{f}(S) &= \sum_{i \in S} (f_i - \phi_{i,t}) + \sum_{i \notin S} (f'_i - \phi_{s,i}) + \sum_C \left[f_C(S \cap C) - \sum_{i \in S, j \in C \setminus S} \phi_{i,j,C} + \sum_{j \in S, i \in C \setminus S} \phi_{i,j,C} \right] \\
&= \sum_{i \in S} f_i + \sum_{i \notin S} f'_i + \sum_C f_C(S \cap C) \\
&\quad - \sum_{i \in S} \left(\phi_{i,t} + \sum_C \sum_{j \in C \setminus S} \phi_{i,j,C} \right) + \sum_{i \notin S} \left(-\phi_{s,i} + \sum_C \sum_{i \in S \cap C} \phi_{i,j,C} \right) \\
&= f(S) - \phi(S, \bar{V} \setminus S) - \phi(\{s\}, V \setminus S) + \phi(V \setminus S, S)
\end{aligned} \tag{5.10}$$

where $\phi(A, B)$ is the total flow from a subset $A \subseteq \bar{V}$ to $B \subseteq \bar{V}$. We will use the fact that $\phi(A \cup B, C) = \phi(A, C) + \phi(B, C)$ when A, B and C are disjoint.

Because flow is conserved at all nodes $i \neq s, t$ we have that the flow into S equals the flow out of S , so $\phi(S, \bar{V} \setminus S) = \phi(\bar{V} \setminus S, S)$, and in particular, the last line above is

$$\begin{aligned}
\bar{f}(S) &= f(S) - \phi(\bar{V} \setminus S, S) - \phi(\{s\}, V \setminus S) + \phi(V \setminus S, S) \\
&= f(S) - \left(\phi(\{s\}, S) + \phi(V \setminus S, S) \right) - \phi(\{s\}, V \setminus S) + \phi(V \setminus S, S) \\
&= f(S) - \phi(\{s\}, V) = f(S) - \nu(\phi)
\end{aligned} \tag{5.11}$$

□

5.1.3 The Max-Flow Min-Cut Theorem for SoS Functions

The key theorem² relating SoS flow and SoS function minimization is a direct analogue of the max-flow min-cut theorem. Recall that in standard max-flow on a graph, a flow is a maximum flow if and only if there are no augmenting paths from s to t . Once we have found this maximum flow, the minimum cut

²All proofs, theorems and lemmas in this section are adapted from [53]

is obtained by taking S^* to be the set of nodes reachable from s along arcs of positive residual capacity (which, by definition, can't include t , otherwise there would be an augmenting path from s to t). In SoS flow, we get a directly analogous theorem.

Given a feasible SoS flow ϕ , define the set of residual arcs A_ϕ to be all arcs a with $\bar{c}_a > 0$. An augmenting path is an $s - t$ path along arcs in A_ϕ . We will say that a feasible flow ϕ^* is maximal if there are no augmenting paths in A_{ϕ^*} .³

Theorem 67. *Let ϕ^* be a maximal flow. Let S^* be the set of all $i \in V$ reachable from s along arcs in A_{ϕ^*} . Then $f(S^*)$ is the minimum value of f over all $S \subseteq V$, and $f(S^*) = \nu(\phi^*)$.*

The simplest proof of this theorem uses the reparameterization result above (Lemma 65). We have mentioned that at a maximal flow ϕ^* , the reparameterization \bar{f} is particularly easy to minimize. In fact, its minimum is the set S^* defined above, which can be found by computing a depth-first search from s in the residual graph A_{ϕ^*} .

Lemma 68. *Let ϕ^* be a maximal flow, and let S^* be the set of i reachable from s along arcs in A_{ϕ^*} . Then*

$$\bar{f}(S^*) = 0 \tag{5.12}$$

Then, the theorem follows immediately from this Lemma, Lemma 65 and Corollary 66, since $f(S) \geq \nu(\phi^*)$ for all sets $S \subseteq V$ and $f(S^*) = \bar{f}(S^*) + \nu(\phi^*) = \nu(\phi^*)$.

³As in standard max-flow, any maximal flow is in fact a maximum flow (i.e., all maximal flows have the same value $\nu(\phi^*)$). Note that this follows immediately from Theorem 67, since all maximal flows have value $f(S^*)$, but to avoid being circular, we'll state the theorem in terms of maximal flows.

Proof of Lemma 68. First, note that we have that $\bar{f}_i = 0$ for $i \in S$, otherwise we would have that t is reachable from some $i \in S$, which would give an augmenting path from s to t passing through i . Similarly, we have $\bar{f}'_i = 0$ for $i \notin S$, otherwise i would be reachable from s along arcs in A_{ϕ^*} .

Now, we just need to show that $\bar{f}_C(S^* \cap C) = 0$ for all cliques C . Fix a C , and let $T = S^* \cap C$. If $T = \emptyset$ or $T = C$ then we're done, since $\bar{f}_C(\emptyset) = \bar{f}_C(C) = 0$. So, we can assume that T and $C \setminus T$ are both nonempty.

Pick any $i \in T$ and $j \in C \setminus T$. We know that j is not reachable from i along arcs of positive residual capacity, so in particular we must have $\bar{c}_{i,j,C} = 0$. Since $\bar{c}_{i,j,C} = \min_{S \subseteq C: i \in S, j \notin S} \bar{f}_C(S)$ there is some $T_{i,j}$ with $\bar{f}_C(T_{i,j}) = 0$ and $i \in T_{i,j}$, $j \notin T_{i,j}$.

Now, let $T_i = \bigcap_{j \in C \setminus T} T_{i,j}$. We have that $i \in T_i$ and each $j \in C \setminus T$ is not in T_i (since $j \notin T_{i,j} \supseteq T_i$). Let $T' = \bigcup_{i \in T} T_i$. We have that $T' = T$, since each $i \in T$ is in T_i , hence also in T' , and each $j \in C \setminus T$ is not in any of the T_i , hence not in T' .

Therefore, we can write $T = \bigcup_{i \in T} \bigcap_{j \in C \setminus T} T_{i,j}$, and $\bar{f}_C(T_{i,j}) = 0$ for all $i \in T$, $j \in C \setminus T$. So, we must have $\bar{f}_C(T) = 0$ since the zero sets of \bar{f}_C form a lattice (i.e., they are closed under intersection and union), by Corollary 38. \square

The key idea of this proof, and the reason that this algorithm only works for submodular functions, is that the zero sets of nonnegative submodular functions form a particular structure called a lattice, meaning they are closed under intersections and unions. In fact, the flow values we're adding and subtracting

are linear functions in S :

$$\begin{aligned}
\sum_{i \in S, j \notin S} \phi_{i,j,C} - \sum_{i \notin S, j \in S} \phi_{i,j,C} &= \sum_{i \in S, j \notin S} \phi_{i,j,C} - \left(\sum_{i \in S, j \in S} \phi_{i,j,C} - \sum_{i \in S, j \in S} \phi_{i,j,C} \right) - \sum_{i \notin S, j \in S} \phi_{i,j,C} \\
&= \sum_{i \in S, j \in C} \phi_{i,j,C} - \sum_{i \in C, j \in S} \phi_{i,j,C} \\
&= \sum_{i \in S} \sum_{j \in C} (\phi_{i,j,C} - \phi_{j,i,C}) \\
&= \sum_{i \in S} \psi_i
\end{aligned} \tag{5.13}$$

where $\psi_i := \sum_{j \in C} (\phi_{i,j,C} - \phi_{j,i,C})$ is the net outflow of node i along arcs in C . Then, we have that $\bar{f}_C(S) = f_C(S) + \psi(S)$, and in particular we have that $\psi \leq f_C$ and $\psi(C) = f_C(C)$ so that ψ is actually a *base* of f_C (see Section 2.3.3).

So, the flow values we're adding and subtracting end up being a search over the base polytope of f_C . An arc $(i, j)_C$ becomes saturated exactly when there's a set $T_{i,j}$ with $i \in T_{i,j}, j \notin T_{i,j}$ which is tight. Then, since the tight sets of \bar{f}_C form a lattice, we can take intersections and unions to get a single, consistent S^* which includes all the nodes reachable from s , and which is itself a tight set.

5.2 IBFS for Submodular Flow

The max-flow min-cut theorem gives a simple algorithm for finding the minimizer of an SoS function — keep track of the current flow and residual graph A_ϕ , and each iteration find an augmenting path from s to t until no more exist. It is easy to show that with integer cost functions this algorithm must terminate in finitely many steps. This augmenting path algorithm was used in Generic Cuts [3], which was the first application of SoS optimization to computer vi-

sion, and the first implementation of SoS flow.

For flow on graphs, the current state of the art for computer vision applications is Incremental Breadth First Search (IBFS) [30]. This algorithm is based on the Boykov-Kolmogorov algorithm of [10], with an additional guarantee of polynomial time complexity. In this section, we show how to modify IBFS to compute maximum SoS flows, giving a fast algorithm for sum-of-submodular optimization for typical computer vision inputs.

5.2.1 IBFS on Graphs

IBFS is an augmenting paths algorithm: at each step, it finds a path from s to t with positive residual capacity, and pushes flow along it. Additionally, each augmenting path found is a shortest s - t path in A_ϕ . To ensure that the paths found are shortest paths, we keep track of distances $d_s(i)$ and $d_t(i)$ from s to i and from i to t , and search trees S and T containing all nodes of distance at most D_s from s or D_t from t respectively. Two invariants are maintained:

- For every i in S , the unique path from s to i in S is a shortest s - i path in A_ϕ .
- For every i in T , the unique path from i to t in T is a shortest i - t path in A_ϕ .

The algorithm proceeds by alternating between forward passes and reverse passes. In a forward pass, we attempt to grow the source tree S by one layer (a reverse pass attempts to grow T , and is symmetric). To grow S , we scan through the vertices i at distance D_s away from s , and examine each out-arc (i, j) that has positive residual capacity. If j is not in S or T , then we add j to S at distance level $D_s + 1$, and with parent i . If $j \in S$ then (i, j) is a back-arc, and is not on the

shortest path from s to j . If j is in T , then we found an augmenting path from s to t via the arc (i, j) , so we can push flow on it.

The operation of pushing flow may saturate some arcs (and cause previously saturated arcs to become unsaturated). If the parent arc of a node i becomes saturated, then i becomes an *orphan*. After each augmentation, we perform an adoption step, where each orphan finds a new parent. The details of the adoption step are similar to the relabel operation of the Push-Relabel algorithm [14], in that we search all potential parent arcs in A_ϕ for the neighbor with the lowest distance label, and make that node our new parent. If this increases the distance $d_s(i)$, then the children of i also become orphans and are recursively adopted as well, to maintain the shortest-path invariant.

5.2.2 Modifying IBFS for SoS Flow

In order to apply IBFS to the SoS flow problem (instead of standard graph-flow), all the basic datastructures still make sense: we have a graph where the arcs a have residual capacities \bar{c}_a , and a maximum flow has been found if and only if there is no longer any augmenting path from s to t .

The main change for the SoS flow problem is that when we increase flow on an edge $(i, j)_C$, instead of just affecting the residual capacity of that arc and the reverse arc, we may also change the residual capacities of other arcs $(i', j')_C$ for $i', j' \in C$. A problematic case would be where $(i', j')_C$ is saturated because a set S has $\bar{f}_C(S) = 0$, with $i' \in S, j' \notin S$. If we push δ units of flow from i to j going into S (meaning $j \in S$ and $i \notin S$) then $\bar{f}_C(S)$ will now be $\delta > 0$, so (i', j') is no longer saturated. If $d(j') > d(i') + 1$ before the push, then we would have created

a shortcut between i' and j' , which would violate our invariants on the trees S and T .

However, the following result ensures that this is not a problem. Let A_ϕ be the set of arcs with residual capacity, according to the current flow.

Lemma 69. *If $(a, b)_c$ was previously saturated, but now has residual capacity as a result of increasing flow along (c, d) , then (1) either $a = d$ or there was an arc $(a, d) \in A_\phi$ and (2) either $b = c$ or there was an arc $(c, b) \in A_\phi$.*

Corollary 70. *Increasing flow on an edge never creates a shortcut between s and i , or from i to t .*

These results are based on [26], we will prove them in the next section.

Corollary 70 ensures that we never create any new shorter s - i or i - t paths not contained in S or T . A push operation may cause some edges to become saturated, but this is the same problem as in the normal max-flow case, and any orphans so created will be fixed in the adoption step. Therefore, all invariants of the IBFS algorithm are maintained, even in the submodular flow case.

The final difference between IBFS and a standard augmenting paths algorithm is the “current arc heuristic”, which is a mechanism for avoiding iterating through all possible potential parents when performing an adoption step. In the case of Submodular Flows, it is also the case that whenever we create new residual arcs we maintain all invariants related to this current arc heuristic, so the same speedup applies here. We cover this heuristic in Section 5.4.

5.2.3 Running Time

The asymptotic complexity of the standard IBFS algorithm is $O(n^2m)$. In the submodular-flow case, we still perform the same number of basic operations. However, note finding residual capacity of an arc $(i, j)_C$ requires minimizing $\bar{f}_C(S)$ for S separating i and j . If $|C| = k$, this can be done in time $O(k^6)$ using [73]. However, for $k \ll n$, it will likely be much more efficient to use the $O(2^k)$ naive algorithm of searching through all values of \bar{f}_C . Overall, we add $O(2^k)$ work at each basic step of IBFS, so if we have m cliques the total runtime is $O(n^2m2^k)$.

This runtime is better than the augmenting paths algorithm of [3] which takes time $O(nm^22^k)$. Additionally, IBFS has been shown to be very fast on typical vision inputs, independent of its asymptotic complexity [30].

5.3 Proof of the “No Shortcuts” Lemma

Lemma 69. *If $(a, b)_C$ was previously saturated, but now has residual capacity as a result of increasing flow along (c, d) , then (1) either $a = d$ or there was an arc $(a, d) \in A_\phi$ and (2) either $b = c$ or there was an arc $(c, b) \in A_\phi$.*

Proof. The flow before the push on (c, d) is denoted by ϕ , and the set of all arcs with residual capacity for flow ϕ is A_ϕ . Recall that when we increase flow on

(c, d) , we change the residual clique functions \bar{f}_C by

$$\bar{f}'_C(S) = \begin{cases} \bar{f}_C(S) - \delta & c \in S, d \notin S \\ \bar{f}_C(S) + \delta & c \notin S, d \in S \\ \bar{f}_C(S) & \text{otherwise} \end{cases} \quad (5.14)$$

Additionally, we defined the residual capacity of the arc $(i, j)_C$ to be

$$c_{i,j,C} = \min_S \{\bar{f}_C(S) \mid i \in S, j \notin S\}. \quad (5.15)$$

We'll say that a set $S \subseteq C$ is saturated if $\bar{f}_C(S) = 0$, and that S separates i from j if $i \in S, j \notin S$. We're interested in which saturated sets separate i from j , so denote the saturated sets by $\mathcal{S}_{i,j}$ defined as

$$\mathcal{S}_{i,j} = \{S \subseteq C \mid i \in S, j \notin S, \bar{f}_C(S) = 0\}. \quad (5.16)$$

By (5.15), $(i, j)_C$ is saturated if and only if $\mathcal{S}_{i,j} \neq \emptyset$. In particular, since $(a, b)_C$ is initially saturated, we know that there's some $S_{a,b} \in \mathcal{S}_{a,b}$.

Now, to prove (1): assume $a \neq d$ (otherwise, we're done). We want to show that the edge (a, d) had nonzero residual capacity in the flow ϕ , so by the above, we need to show that $\mathcal{S}_{a,d} = \emptyset$. Assume by way of contradiction that there exists $S_{a,d} \in \mathcal{S}_{a,d}$.

Let $S_a = S_{a,b} \cap S_{a,d}$. We know that $a \in S_a$ since it's in both $S_{a,b}$ and $S_{a,d}$. We also know that both b and d are not in S_a , since $b \notin S_{a,d}$ and $d \notin S_{a,b}$. Finally, S_a is the intersection of two saturated sets, so it must also be saturated, by Corollary 38 (which says that the zero-valued sets of \bar{f} form a lattice).

Therefore, S_a is a set containing a but not b or d , and with $\bar{f}_C(S_a) = 0$. When we change flow on (c, d) , according to (5.14), we only increase the capacity of

sets containing d . So after changing ϕ , we still have $\bar{f}_C(S_a) = 0$. But then, S_a separates a from b , so $(a, b)_C$ continues to be saturated, a contradiction.

To prove (2), we use a similar argument. We have $b \neq c$ or else we're done. Assume by way of contradiction that there exists $S_{c,b} \in \mathcal{S}_{c,b}$. Let $S_c = S_{c,b} \cup S_{a,b}$. We know that $a, c \in S_c$ and $b \notin S_c$. Furthermore, since S_c is the union of two zero-valued sets, it also has $\bar{f}_C(S_c) = 0$.

But then, since $c \in S_c$, we know that the capacity of S_c doesn't increase, and therefore after changing ϕ it continues to be a saturated set. But since $a \in S_c$, $b \notin S_c$ this means that $(a, b)_C$ continues to be saturated, a contradiction.

□

Corollary 70. *Increasing flow on an edge never creates a shortcut between s and i , or from i to t .*

Proof. Assume we just increased flow on arc $(c, d)_C$, causing $(a, b)_C$ to have positive residual capacity. We'll consider the case where $c \in S$, the case where $d \in T$ follows by symmetry. Because we increased flow on $(c, d)_C$, it is along the shortest path from s to d , so $d_s(d) = d_s(c) + 1$.

We know that $(c, d)_C$ is either a tree arc, or an arc from S to T and hence $d_s(d) = d_s(c) + 1$. Since either $d = a$ or $(a, d) \in A_\phi$, we have $d_s(d) \leq d_s(a) + 1$. Similarly, since either $c = b$ or $(c, b) \in A_\phi$ we have $d_s(b) \leq d_s(c) + 1$. Putting these inequalities together, we get that

$$d_s(b) \leq d_s(c) + 1 = d_s(d) \leq d_s(a) + 1 \quad (5.17)$$

Therefore, when we cause (a, b) to become unsaturated, we don't create a

new shortest path from s to b . The argument that we don't create a new shortest path from a to t is symmetric. \square

5.4 The Current Arc Heuristic

Finally, we will describe the “current arc heuristic” of IBFS, and show how its invariants still hold in the submodular flow case. We will discuss the current-arc mechanism for the source tree S , the case regarding T is symmetric. Some useful terminology: an arc (u, v) is admissible if $d_s(v) = d_s(u) + 1$ and $(u, v) \in A_\phi$. Only admissible arcs can be tree arcs.

For every $v \in S$, we maintain a list of potential parent arcs (u, v) in an unspecified order, denoted $(u, v) < (u', v)$. The *parent node* is the parent of v in the tree S , denoted $p(v)$. At any point in the algorithm, the *current arc* is the arc from this list which is currently the parent arc of v (i.e., the current arc is always $(p(v), v)$). We maintain the invariant that all arcs before the current arc (according to the order $<$) are not admissible. Therefore, when searching for a new parent for v in an adoption step, we don't have to scan the entire list of potential arcs, but can instead increment the current arc until we find an admissible arc. If we get to the end of the list, we know that all arcs into v are currently inadmissible, so we must increase the label of v . This scheme allows us to “charge” the scanning of edges to the relabels of v , and there are only $O(n)$ relabels per vertex.

In order to make our scheme work for the submodular flow case, we specify a particular order of in-arcs for each v . Put a linear ordering $<$ on V , and order the arcs so that if $u < w$ then $(u, v) < (w, v)$. We use the same linear ordering for doing our breadth-first search, so we scan through the nodes at distance D_s in

sorted order, according to $<$.

We need to maintain the invariant that arcs before the current arc $(p(v), v)$ are not admissible. Equivalently, if $w < p(v)$ then (w, v) is not an admissible arc.

When a vertex v first enters S , IBFS sets the current arc to the first arc in the list (because we're scanning through nodes at distance D_s in $<$ -sorted order), so the invariant is initially true. So, assume that (u, v) is the current arc of v at a particular stage in the algorithm, and consider all the ways we could violate the invariant. The first two cases are already covered by the proof of correctness of the standard IBFS algorithm, but we repeat them briefly for completeness.

- We could change the current arc (i.e., change the parent of v). But we only do this during an adoption step, when we know that (u, v) is inadmissible. All arcs before (u, v) are also inadmissible, and we scan forward through the list till we find an admissible (w, v) with $u < w$ (or reach the end), which maintains the invariant.
- We could relabel a vertex w with $w < u$ and $(w, v) \in A_\phi$. But relabels only ever increase the label of w . If $d(w) < d(v)$ then $d(w) = d(v) - 1$ (if $d(w) < d(v) - 1$ then (w, v) would already be on the shortest path to v). So, increasing the label of w makes (w, v) inadmissible.
- We could change the flow ϕ such that an arc (w, v) which was previously saturated becomes unsaturated, and $d(v) = d(w) + 1$.

This is only a problem if $w < u$, as the invariant has nothing to say about creating admissible arcs after the current arc. This is illustrated in Figure 5.1 (Left). By Lemma 2.3, the only way we could cause (w, v) to become unsaturated is by pushing on some arc (x, y) such that (1) $w = y$ or

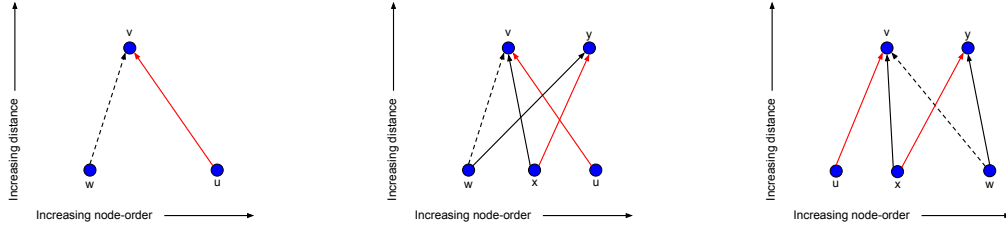


Figure 5.1: Illustrations of the flow network regarding the current arc heuristic. Saturated edges are dotted while unsaturated edges are solid, parent arcs are colored red. The linear order in nodes increases from left to right. (Left) A potential failure of the current arc heuristic. The node w is less than u and pushing on some other arc causes (w, v) to become unsaturated. (Center) A potential arrangement of the nodes, where (x, y) is the node whose flow is increasing. Note that this configuration is impossible because the parent arc of y is (x, y) which comes after (w, y) . Similarly the parent arc of v is (u, v) which comes after (x, v) . (Right) the only possible configuration of these nodes. In this case, we don't create a violation of the current arc heuristic, since the new edge created has $u < w$.

$(w, y) \in A_\phi$ and (2) $x = v$ or $(x, v) \in A_\phi$. Recall, from our proof of Corollary 70 that this implies $d(v) \leq d(x) + 1$ and $d(y) \leq d(w) + 1$.

Since we're pushing on (x, y) we know that (x, y) is a tree-arc, so $x = p(y)$ and $d_s(y) = d_s(x) + 1$. If $w = y$ then $d(w) = d(y) = d(x) + 1 \geq d(v)$ so we don't create an admissible arc. Similarly, if $x = v$ then $d(v) = d(x) = d(y) - 1 \leq d(w)$ so we again don't create an admissible arc.

Since neither $w = y$ or $x = v$, we're in the case of Figure 5.1 (Center), where there are arcs (x, y) , (w, y) and (x, v) , all of which are in A_ϕ . If we've created a new admissible arc (w, v) , then $d(v) = d(w) + 1$, and hence $d(v) = d(w) + 1 \geq d(y) = d(x) + 1 \geq d(v)$ so $d(v) = d(y) = d(w) + 1 = d(x) + 1$. Therefore, each of the three arcs (x, y) , (w, y) and (x, v) are also admissible.

Since $x = p(y)$ we know (since the invariant holds before we change ϕ) that

$x \leq w$, as (w, y) is admissible. Similarly, since (u, v) is the current arc of v , we must have that $u \leq x$ since (x, v) is an admissible arc. Therefore, $u \leq x \leq w$, and hence whenever we create a new admissible arc, we create one after the current arc of v , so the invariant is maintained.

Since the current arc invariants are maintained, all the arguments regarding the runtime of IBFS hold in the submodular flow case, and we get an overall number of $O(n^2m)$ basic steps, for a total of $O(n^2m2^k)$ total time.

CHAPTER 6

SUBMODULAR UPPER BOUNDS FOR HIGHER ORDER ENERGY FUNCTIONS

6.1 Introduction

Now that we have a tool, Sum-of-Submodular flow, for optimizing binary submodular MRFs, we want to apply in a graph-cuts algorithm for more general higher-order MRFs. The key challenge is that most higher-order vision MRFs are not submodular, including the denoising and stereo applications that popularized higher-order MRFs [77, 102]. Our approach for optimizing these non-submodular functions is to instead find a submodular function which is close to the original function, and then exactly optimize that submodular proxy. QPBO [8, 51] takes this approach, as does its generalization GRD [43].

Given an arbitrary binary function f , the natural choice of submodular proxy g is an upper bound on f . Since $g \geq f$, when we make g small we also make f small. We will call g a *submodular upper bound*. Experimentally, minimizing submodular upper bounds has been successful at optimizing both pairwise binary functions with the local search algorithm LSA-AUX [31], as well higher-order multilabel problems with Auxiliary Cuts [6] and SoSPD [22].

The main contribution of this chapter is to derive a principled submodular upper bound for higher-order MRFs. We show that we can minimize the distance between the submodular upper bound and the original function, and we give fast algorithms for finding approximately optimal upper bounds in practice.

6.2 Background and Related Work

For this chapter, the most useful definition of submodularity is the following equivalent condition from Theorem 32: f is submodular if

$$f(S) + f(S + i + j) \leq f(S + i) + f(S + j) \quad (6.1)$$

for all $S \subseteq V$ and $i, j \notin S$.

For higher-order functions, any submodular function can be minimized in $O(n^6)$ time [73]; however, this is impractical for vision-sized inputs. A more practical alternative is sum-of-submodular optimization, which fits between graph cuts and submodular optimization in complexity. The Submodular IBFS algorithm (Chapter 5) in particular is well-suited to vision inputs, as it generalizes the popular Boykov-Kolmogorov¹ algorithm [10] to higher-order inputs, and for fixed-sized cliques runs in worst-case time $O(n^2m)$ for m cliques and n variables (the same complexity as IBFS for pairwise inputs). However, it scales as $O(2^{|C|})$ as the clique size grows, so we will henceforth only consider higher-order inputs with a small constant for the size of the cliques $|C|$. Currently, all known methods for optimizing general higher-order functions have this exponential (or at least $O(|C|^6)$) dependence on the size of the cliques. Faster methods exist given for certain special cases of energy functions (such as robust- P^n [49], and other structured cost functions), but cannot handle arbitrary energies, as the current method does.

For non-binary problems, graph cuts methods typically reduce the solution to a series of binary sub-problems, using move-making algorithms such

¹The Boykov-Kolmogorov flow algorithm and IBFS [30] are very similar — the latter makes a small tweak to a subroutine which allows proving an $O(n^2m)$ runtime. Despite this, the implementation of BK flow remains popular within the vision community.

as alpha-expansion [11] and fusion moves [66], as well as more sophisticated primal dual algorithms such as FastPD [58], and its higher-order generalization SoSPD [22].

The major restriction in the above is that the binary problems are required to be submodular. If the functions are not submodular, then the problem is NP-hard [52]. Yet many optimization problems encountered in practice are not submodular, so we need a submodular function whose optimum is close to the original.

Submodular upper bounds have been used before in several different methods to approximate arbitrary functions. The FastPD algorithm [58] uses a very simple upper bound — if, during a particular expansion move, the cut-cost of an edge would be negative, the variant PD3_a simply truncates that cost to 0, ensuring the expansion move problem is submodular (this approach originated in [80]). A more sophisticated upper bound was employed in the LSA-AUX method [31] which found a series of upper bounds g^t , each of which has $g^t(\mathbf{x}^t) = f(\mathbf{x}^t)$ at the current point \mathbf{x}^t . Each move solves a pairwise submodular minimization problem $\mathbf{x}^{t+1} = \arg \min g^t(\mathbf{x})$ with graph cuts, giving a fast local-search method. However, for pairwise functions, submodular upper bounds are particularly simple: either the edge term $f_{i,j}$ is already submodular (in which case no upper bound is necessary) or it is supermodular, in which case the best upper bound is a linear function.

For higher-order functions, Auxiliary Cuts [6] uses convexity and other properties of image functionals to compute upper bounds which are pairwise submodular functions. These upper bounds are iteratively minimized and updated similarly to LSA-AUX. The Pseudo-Bound method [92] extends this idea

by considering a parameterized family of functions which include a pairwise submodular upper bound, and finding all minimizers of the entire family using parametric max-flow — by looking at all minimizers, a greater decrease in energy per iteration is obtained. Note that for all these methods, the upper bounds are all pairwise functions, even if the functions they approximate are higher-order.

Our approach is based on a linear program for minimizing the distance between the submodular upper bound and the original function. We give new approximations to this LP that have nice theoretical properties, and which perform well in practice. For multilabel problems, these upper bounds allow approximate optimization of expansion and fusion moves, even when the move-making binary-subproblem is non-submodular. We will show that by employing in our upper bounds in a Fusion-moves framework, better energy optimization is obtained.

6.2.1 Notation

Some notation we will use throughout the rest of the chapter: Recall that for binary problems, we are identifying set functions $f(S)$ with the vector notation $f(\mathbf{x})$, $\mathbf{x} \in \{0, 1\}^n$. For a set S , and element i , we will write $S + i$ for $S \cup \{i\}$. For a set of coefficients a_i for $i \in V$, we let $a(S) = \sum_{i \in S} a_i$. Note that a is a linear function. For a set of coefficients $b_{i,j}$ on edges $\{i, j\}$, we will write

$$b(S) = \sum_{i < j, i, j \in S} b_{i,j} \tag{6.2}$$

Note that b is a quadratic function. Whenever we use a pair of indices i, j , this is always an unordered pair, so $b_{i,j} = b_{j,i}$ and $\delta_{S,i,j} = \delta_{S,j,i}$. For two set functions

$f, g : 2^V \rightarrow \mathbb{R}$ we will define the 1-norm and ∞ -norm distance between them by treating the functions as length $2^{|V|}$ vectors, i.e., $\|g - f\|_1 = \sum_S |g(S) - f(S)|$ and $\|g - f\|_\infty = \max_S |g(S) - f(S)|$.

6.3 Submodular Upper Bounds

Recall that a function is submodular if it satisfies (6.1) for all $S \subseteq V$ and $i, j \notin S$. Given this definition, we can write an optimization problem minimizing the p -norm distance between an arbitrary function f over all submodular upper bounds g .

$$\begin{aligned}
& \min_g \|g - f\|_p \\
& \text{s.t. } g(S) \geq f(S) \quad \forall S \subseteq V \\
& \quad g(S) + g(S + i + j) \leq g(S + i) + g(S + j) \quad \forall S \subseteq V, i, j \notin S
\end{aligned} \tag{6.3}$$

For $p = 1$ and $p = \infty$ the problem (6.3) is a linear program. There are $2^{|C|}$ variables, however for small clique size this can be solved exactly using off-the-shelf LP solvers. Note that small clique sizes is also the restriction for when sum-of-submodular flow is tractable, so this is not an additional restriction. We do apply this upper bound clique-by-clique, so that the final sum-of-submodular function g is the sum $g = \sum_C g_C$ of upper bounds for each clique, $g_C \geq f_C$.

It's worth noting that other distance measures than the 1- and ∞ -norm are possible here. For example, we also considered the 2-norm — however, this leads to a quadratic program in the objective of (6.3), which is harder to solve. Experimentally, we found that the 2 norm was outperformed by the ∞ -norm on

the examples in Section 8.2, while taking longer to compute.

One reason for choosing the ∞ -norm is that we can prove a global approximation bound on the minimization problem $\min_{\mathbf{x}} f(\mathbf{x})$. Let $k_C = \|g_C - f_C\|_\infty$ be the ∞ -norm objective for (6.3) for each clique C . Then, we always have $f_C(\mathbf{x}_C) \geq g_C(\mathbf{x}_C) - k_C$ for any \mathbf{x}_C , and hence $f(\mathbf{x}^*) \geq g(\mathbf{x}^*) - \sum_C k_C$ for the minimizing assignment \mathbf{x}^* . That is, $\sum_C k_C$ is an additive approximation bound relating the minimum of g and the minimum of the original function f . In practice, this is a very weak bound; however, it does argue that improving the ∞ -norm distance between f and g will result in better energy after minimizing g .

Finally, we'll note that the upper bound for pairwise MRFs proposed in the LSA-AUX method of [31] is a special case of (6.3). This upper bound is obtained by taking each quadratic edge term $f_{i,j}(x_i, x_j) = b_{i,j}x_i x_j$ of the objective: if $b_{i,j} \leq 0$ then the term is submodular, in which case the upper bound is $g_{i,j} = f_{i,j}$. If $b_{i,j} > 0$ then the term is supermodular, in which case they give a linear upper bound $g_{i,j}(x_i, x_j) = b_{i,j} \frac{x_i + x_j}{2}$. It is simple to check that this linear upper bound minimizes (6.3) for both $p = 1$ and $p = \infty$. Essentially, in the pairwise case, there aren't enough degrees of freedom, so all nontrivial submodular upper bounds are linear functions. This is not the case for cliques with $|C| > 2$.

6.4 Upper Bound Approximations

In practice, even though we can solve (6.3) using linear programming, it is much more efficient to find approximate solutions (in some of our experiments, optimization using the LP upper bounds took upwards of an hour). In this section, we will present 2 alternate submodular upper bounds, both of which minimize

the 1-norm objective of (6.3) under some additional constraints, and both of which are much easier to compute.

Our intuition for these upper bounds is to first consider the question: what simple families of submodular functions are there? In particular, there are two classes of functions which can easily be shown to be submodular. The first are quadratic (pairwise) functions, in which every quadratic term has non-positive coefficient. The second family are functions of the form $\phi(S) = h(|S|)$ where h is a concave function.

6.4.1 The Iterative Heuristic of SoSPD

The first upper bound method for SoS functions was the heuristic employed by the original paper for the SoSPD [22] algorithm (Chapter 7). The basic intuition is to first look at the equations defining submodularity, $f(S) + f(S + i + j) \leq f(S + i) + f(S + j)$. If this inequality is violated, we either lower $f(S)$, $f(S + i + j)$ or raise $f(S + i)$, $f(S + j)$. Since we're finding an upper bound, we can only increase the values. Thus, our only options are to increase $f(S + i)$ or $f(S + j)$.

The SoSPD heuristic simply iterates through all the sets in decreasing size, and increases both $f(S + i)$ and $f(S + j)$ by the amount needed to fix the violated inequality. However, increasing $f(S + i)$ may cause some other inequality to be violated (e.g., $f(S - k) + f((S - k) + i + k) \leq f((S - k) + i) + f((S - k) + k)$ for some k). Therefore, this method needs to repeatedly iterate through all the sets S until all inequalities are satisfied.

The specifics of this method are given in the supplementary material of [22];

however, it was primarily intended as a simple heuristic for a subroutine of the SoSPD algorithm, and is both theoretically and experimentally inferior to the principled lower bounds below. This method is guaranteed to converge, however there are no results concerning how close to the original function f this upper bound will be.

6.4.2 Quadratic-Based Submodular Upper Bounds

Consider a quadratic function $q(\mathbf{x}) = \sum_i a_i x_i + \sum_{i < j} b_{i,j} x_i x_j$. In set notation, $q(S) = a(S) + b(S)$, and q is submodular if and only if $b_{i,j} \leq 0$ for all i, j . Our goal is to find a submodular quadratic function q such that $g_q(S) := f(S) + q(S)$ is submodular and $g_q \geq f$.

To do so, define $\delta_{S,i,j} = f(S) + f(S + i + j) - f(S + i) - f(S + j)$. That is, $\delta_{S,i,j}$ is the amount by which the submodularity inequality for S, i, j is violated (possibly negative if the inequality is satisfied). For i, j , define $\Delta_{i,j} = \max\{\max_{S: i, j \notin S} \delta_{S,i,j}, 0\}$, the maximum violation over all constraints containing i, j .

Construct a quadratic submodular function q^* by setting $b_{i,j}^* = -\Delta_{i,j}$ for all i, j and $a_i^* = \frac{1}{2} \sum_j \Delta_{i,j}$ for all i . We now show that q^* is actually the best q which makes g_q a submodular upper bound to f , under the 1-norm. More specifically, consider the program

$$\begin{aligned}
& \min_q \|g_q - f\|_1 \\
& \text{s.t. } g_q = f + q \\
& g_q \geq f \\
& g_q \text{ and } q \text{ are submodular}
\end{aligned} \tag{6.4}$$

Theorem 71. q^* is a minimizer of (6.4).

Before we prove Theorem 71, we will note that even though q is quadratic, g_q may not be. In particular, for the function f of three variables, $f(\mathbf{x}) = 2x_1x_2 - x_1x_2x_3$, we have $q^*(\mathbf{x}) = x_1 + x_2 - 2x_1x_2$ and $g_{q^*}(\mathbf{x}) = x_1 + x_2 - x_1x_2x_3$, which is submodular, but not quadratic.

Also, it's worth noting that q^* is an approximate solution to (6.3) in that we are minimizing over a smaller set, by restricting g to be $f + q$ for a submodular, quadratic q . However, we can compute q^* quickly, by iterating first over all pairs (i, j) and then over all S with $i, j \notin S$ and computing $\delta_{S,i,j}$ for each. For small clique sizes C this operation is very efficient (though it is $O(2^{|C|})$ as $|C|$ grows).

We now prove Theorem 71. First, we show that the constraints of (6.4) can be re-written as inequalities on the coefficients a and b of q .

Proposition 72. (6.4) is equivalent to

$$\min_q \{\|q\|_1 : q \geq 0, b_{i,j} \leq -\Delta_{i,j} \forall i, j\} \quad (6.5)$$

Proof. The only non-trivial part of the equivalence is showing we can replace the submodularity constraints with $b_{i,j} \leq -\Delta_{i,j}$ for all i, j . So, begin with the inequalities defining submodularity for g_q : For every S and $i, j \notin S$ we have a constraint

$$g_q(S) + g_q(S + i + j) - g_q(S + i) - g_q(S + j) \leq 0. \quad (6.6)$$

Substituting in $g_q = f + q$, and recalling the definition of $\delta_{S,i,j}$ we have

$$\begin{aligned}
0 &\geq g_q(S) + g_q(S + i + j) - g_q(S + i) - g_q(S + j) \\
&= f(S) + f(S + i + j) - f(S + i) - f(S + j) \\
&\quad + q(S) + q(S + i + j) - q(S + i) - q(S + j) \\
&= \delta_{S,i,j} + q(S) + q(S + i + j) - q(S + i) - q(S + j)
\end{aligned} \tag{6.7}$$

Then, we can expand out q to get

$$\begin{aligned}
&\delta_{S,i,j} + q(S) + q(S + i + j) - q(S + i) - q(S + j) \\
&= \delta_{S,i,j} + a(S) + a(S + i + j) - a(S + i) - a(S + j) \\
&\quad + b(S) + b(S + i + j) - b(S + i) - b(S + j) \\
&= \delta_{S,i,j} + b_{i,j}
\end{aligned} \tag{6.8}$$

Therefore, g_q is submodular if and only if $b_{i,j} \leq -\delta_{S,i,j}$ for all S with $i, j \notin S$. As for the constraint that q is submodular, this happens if and only if $b_{i,j} \leq 0$. Therefore, we have that all these constraints can be replaced by $b_{i,j} \leq -\Delta_{i,j}$ for all i, j . \square

Since $b_{i,j}^* = -\Delta_{i,j}$, to show that q^* is feasible for (6.5) we just need to show that $q^* \geq 0$. Note that we can rearrange q^* using vector notation as follows

$$q^*(\mathbf{x}) = \sum_i a_i^* x_i + \sum_{i < j} b_{i,j}^* x_i x_j \tag{6.9}$$

$$= \sum_i \left(\frac{1}{2} \sum_j \Delta_{i,j} \right) x_i + \sum_{i < j} -\Delta_{i,j} x_i x_j \tag{6.10}$$

$$= \sum_{i < j} \left(\frac{x_i + x_j}{2} - x_i x_j \right) \Delta_{i,j} \tag{6.11}$$

Then, we can check that for the four assignments of $x_i, x_j \in \{0, 1\}$ that $\frac{x_i + x_j}{2} - x_i x_j \geq 0$, so $q^* \geq 0$, and hence q^* is feasible for (6.5).

Finally, to prove Theorem 71 we look at the objective $\|g - f\|_1 = \|q\|_1$.

$$\|q\|_1 = \sum_S q(S) = \sum_S \sum_{i \in S} a_i + \sum_S \sum_{\substack{i, j \in S \\ i < j}} b_{i,j} \quad (6.12)$$

$$= \sum_i \sum_{S: i \in S} a_i + \sum_{i < j} \sum_{S: i, j \in S} b_{i,j} \quad (6.13)$$

$$= 2^{|C|-1} \sum_i a_i + 2^{|C|-2} \sum_{i < j} b_{i,j} \quad (6.14)$$

Since for feasible q we have $g_q(C) - f(C) \geq 0$ we must have $q(C) \geq 0$, and hence $\sum_i a_i \geq -\sum_{i < j} b_{i,j}$. Therefore, we get that $\|g_q - f\|_1 \geq -2^{|C|-2} \sum_{i < j} b_{i,j} \geq 2^{|C|-2} \sum_{i < j} \Delta_{i,j}$. On the other hand

$$\begin{aligned} \|q^*\|_1 &= 2^{|C|-1} \sum_i a_i^* + 2^{|C|-2} \sum_{i < j} b_{i,j}^* \\ &= 2^{|C|-1} \sum_i \left(\frac{1}{2} \sum_j \Delta_{i,j} \right) + 2^{|C|-2} \sum_{i < j} -\Delta_{i,j} \\ &= 2^{|C|-2} \sum_{i < j} \Delta_{i,j} \end{aligned} \quad (6.15)$$

so q^* is optimal.

6.4.3 Cardinality-Based Submodular Upper Bounds

In this section, we will focus another family of functions, the concave cardinality-based function, $\phi(S) = h(|S|)$ where h is concave. To motivate this, recall that such a ϕ is submodular if and only if h is concave. The goal of this section is to find a concave cardinality-based function ϕ such that $g_\phi(S) := f(S) + \phi(S)$ is submodular and $g_\phi \geq f$. We are going to solve the following program:

$$\begin{aligned}
& \min_{\phi} \|g_{\phi} - f\|_p \\
& \text{s.t. } g_{\phi} = f + \phi \\
& g_{\phi} \geq f, \\
& g_{\phi} \text{ and } \phi \text{ are submodular.}
\end{aligned} \tag{6.16}$$

Let's introduce $n = |C|$, and $\psi_k = h(k)$ for $k = 0, 1, \dots, n$ as a shorthand, so that $\phi(S) = h(|S|) = \psi_{|S|}$. We also let $\Delta_k = \max\{\max_{|S|=k-1, i, j \notin S} \delta_{S,i,j}, 0\}$ for $k = 1, \dots, n-1$, which is the maximum submodularity violation over all sets of size $k-1$.² Note again that $\delta_{S,i,j}$ might be negative but Δ_k is always non-negative. Now, we can rewrite (6.16) in an equivalent form:

$$\begin{aligned}
& \min_{\psi} \|\psi\|_p \\
& \text{s.t. } \psi \geq 0, \\
& 2\psi_k \geq \psi_{k-1} + \psi_{k+1} + \Delta_k, \quad k = 1, \dots, n-1
\end{aligned} \tag{6.17}$$

where we define

$$\|\psi\|_1 := \sum_{k=0}^n \binom{n}{k} \psi_k \quad \|\psi\|_{\infty} = \max_k \psi_k \tag{6.18}$$

Lemma 73. (6.16) and (6.17) are equivalent.

Proof. Since $g(S) - f(S) = \psi_{|S|}$, the objective is $\|\psi\|_p = \|g_{\phi} - f\|_p$ by how we've just defined $\|\psi\|_p$. The constraint $\psi \geq 0$ is equivalent to $g_{\phi} \geq f$.

Next, for g_{ϕ} to be submodular, we require

$$f(S + i) + \psi_{|S|+1} + f(S + j) + \psi_{|S|+1} \geq f(S) + \psi_{|S|} + f(S + i + j) + \psi_{|S|+2} \tag{6.19}$$

²Recall that $\delta_{S,i,j}$ is defined as in Section 6.4.2, and gives the violation of the submodular constraint for S, i, j .

for all $S, i, j \notin S$. Rearranging this, we get

$$\begin{aligned}
& 2\psi_{|S|+1} - \psi_{|S|} - \psi_{|S|+2} \\
& \geq f(S) + f(S + i + j) - f(S + i) - f(S + j) \\
& = \delta_{S,i,j}
\end{aligned} \tag{6.20}$$

This must hold for all S and $i, j \notin S$. Equivalently, for every k (where $k = |S| + 1$) we have $2\psi_k - \psi_{k-1} - \psi_{k+1} \geq \max_{S': |S'|=k-1, i, j \notin S'} \delta_{S',i,j}$. Also note that since h is concave we always have $2\psi_k - \psi_{k-1} - \psi_{k+1} \geq 0$. Therefore, the submodularity of g_ϕ is equivalent to the constraints $2\psi_k - \psi_{k-1} - \psi_{k+1} \geq \Delta_k$ for $k = 1, \dots, n-1$. \square

Since $2\psi_k - \psi_{k-1} - \psi_{k+1}$ is widely referred as the *discrete Laplacian operator*, we will refer these constraints as Laplacian constraints. Furthermore, if we require all the inequalities in the Laplacian constraints to be satisfied with equality, these are called the *inhomogeneous Laplacian equations*. Our algorithm to find the cardinality-based upper bound is to solve these Laplacian equations, in a procedure detailed below. We will show later in this section that this algorithm gives us a feasible submodular upper bound. Additionally, ψ^* is optimal under the 1-norm and a 2-approximation to (6.16) under the ∞ -norm.

Without loss of generality, we can assume $\psi_0 = \psi_n = 0$ — since ψ_0 and ψ_n only appear in the RHS of the Laplacian constraints, for any feasible solution ψ , we could decrease ψ_0, ψ_n to get $\psi_0 = \psi_n = 0$ and none of the constraints will be violated and we decrease the objective value.

After fixing ψ_0 and ψ_n to be 0, there are $n-1$ variables $\psi_1, \dots, \psi_{n-1}$ and $n-1$ Laplacian constraints remaining. We can uniquely solve the corresponding Laplacian equations since the coefficient matrix has full rank. Also note that this is very efficient to solve — since the coefficient matrix M is tridiagonal.

onal, we can easily factor it into an LU decomposition and using forward and backward substitution, solve for $M\psi = \Delta$ in $O(n)$ time. The bottleneck of the cardinality-based upper bound computation is still the computation of $\delta_{S,i,j}$ and Δ_k , which takes exponential time in the clique size. The following lemma proves the correctness of the algorithm.

Lemma 74. *Solving the Laplacian matrix gives us a feasible solution for (6.17), where we define $\psi^* = M^{-1}\Delta$.*

Proof. Clearly, all the Laplacian constraints are satisfied since we treat them as equalities and solve the linear system for a point which makes all of them satisfied simultaneously. To show non-negativity, it's straightforward to show that the inverse matrix of Laplacian coefficient matrix is a positive matrix; for completeness, we include this in Appendix B. Our definition of Δ_k is non-negative, hence our solution $\psi^* = M^{-1}\Delta$ is a non-negative matrix times a non-negative vector hence also non-negative. \square

Now, let's show some optimality properties of the cardinality-based upper bound. The following lemma is the key fact for the analysis.

Lemma 75. *For $\forall k \leq \lceil \frac{n}{2} \rceil$, all feasible ψ have $\psi_k + \psi_{n-k} \geq L_k$, where $L_0 = 0$ and $L_k = L_{k-1} + \sum_{i=k}^{n-k} \Delta_i$. Solving the Laplacian equations gives us the ψ which simultaneously minimizes the quantity $\psi_k + \psi_{n-k}$ for $\forall k$, meaning $\psi_k^* + \psi_{n-k}^* = L_k$.*

Proof. We prove this by induction on k . The base case $k = 0$ is trivial since we enforce $\psi_0 = \psi_n = 0$ in our algorithm, and the non-negativity of ψ ensures that $\psi_0 + \psi_n$ reaches its lower bound $L_0 = 0$.

For $k \geq 1$, we can sum up the k -th to the $(n - k)$ -th Laplacian constraints, to get

$$\sum_{l=k}^{n-k} (2\psi_l - \psi_{l-1} - \psi_{l+1}) \geq \sum_{l=k}^{n-k} \Delta_l \quad (6.21)$$

and hence

$$\begin{aligned} \sum_{l=k}^{n-k} \Delta_l &\leq \sum_{l=k}^{n-k} (\psi_l - \psi_{l-1}) + \sum_{l=k}^{n-k} (\psi_l - \psi_{l+1}) \\ &= \psi_{n-k} - \psi_{k-1} - \psi_{n-k+1} + \psi_k \end{aligned} \quad (6.22)$$

and rearranging, we have

$$\psi_k + \psi_{n-k} \geq \psi_{k-1} + \psi_{n-k+1} + \sum_{i=k}^{n-k} \Delta_i \geq L_{k-1} + \sum_{i=k}^{n-k} \Delta_i = L_k \quad (6.23)$$

Since we get ψ 's from solving the Laplacian equations, all the above inequalities hold with equality. So inductively assuming $\psi_{k-1}^* + \psi_{n-k+1}^* = L_{k-1}$ we have

$$\psi_k^* + \psi_{n-k}^* = \psi_{k-1}^* + \psi_{n-k+1}^* + \sum_{i=k}^{n-k} \Delta_i = L_k \quad (6.24)$$

□

Theorem 76. *The cardinality-based upper bound is optimal under the $p = 1$ version of (6.17).*

Proof. We have the 1-norm objective of (6.17) to be $\sum_{k=0}^n \binom{n}{k} \psi_k$. Since the binomial coefficients are symmetric, this objective function is a positive linear combinations of $\psi_k + \psi_{n-k}$. For odd n we have

$$\|\psi\|_1 := \sum_{k=0}^n \binom{n}{k} \psi_k = \sum_{k=0}^{(n-1)/2} \binom{n}{k} (\psi_k + \psi_{n-k}) \quad (6.25)$$

and for even n :

$$\begin{aligned} \|\psi\|_1 &= \sum_{k=0}^{n/2-1} \binom{n}{k} (\psi_k + \psi_{n-k}) \\ &\quad + \frac{1}{2} \binom{n}{n/2} (\psi_{n/2} + \psi_{n/2}) \end{aligned} \quad (6.26)$$

Since we separately minimize each sum $\psi_k + \psi_{n-k}$, due to Lemma 75, we minimize the whole objective as well. □

Theorem 77. *The cardinality-based upper bound gives a 2-approximation under the $p = \infty$ version of (6.17).*

Proof. Under the ∞ -norm, the objective function in (6.17) is $\max_k \psi_k$. In Lemma 75, we have established the lower bound for $\psi_k + \psi_{n-k} \geq L_k$ so that $\max\{\psi_k, \psi_{n-k}\} \geq \frac{L_k}{2}$. Hence, the minimum of (6.17) is at least $\max_k \frac{L_k}{2}$. We also know our choice of ψ^* has $\psi_k^* + \psi_{n-k}^* = L_k$ for each k and all the ψ_k are non-negative, i.e., in the worst case, our algorithm can give us a feasible solution with objective value $\max_k L_k$. Therefore, our algorithm is a 2-approx under ∞ -norm. \square

Additionally, we can prove the following general approximation ratio for arbitrary p -norm ($p \geq 1$), which contains the previous two theorems as special cases. The proof for the general case is analogous to the proof for the 1-norm, and we will defer it to Appendix C

Theorem 78. *The cardinality-based upper bound gives a $2^{(1-\frac{1}{p})}$ -approximation for (6.17).*

r

CHAPTER 7

A PRIMAL-DUAL ALGORITHM FOR HIGHER-ORDER MULTILABEL MARKOV RANDOM FIELDS

Now that we have adapted the state-of-the-art flow algorithm for vision problems to solve higher-order submodular MRFs, and also shown how to handle arbitrary non-submodular binary MRFs using upper bounds, we can turn to the problem of handling multi-label problems. In this chapter we propose a new primal-dual energy minimization method for arbitrary higher-order multilabel MRFs. Primal-dual methods provide guaranteed approximation bounds, and can exploit information in the dual variables to improve their efficiency. Our algorithm generalizes the PD3 [57] technique for first-order MRFs, and relies on the SoS IBFS algorithm of Chapter 5 to optimize the binary MRFs at each step. We provide approximation bounds similar to PD3 [57], and the method is fast in practice. It can optimize non-submodular MRFs, and additionally can incorporate problem-specific knowledge in the form of fusion proposals.

7.1 Higher-order Multi-label MRFs

In multi-label problems, we now allow the label set \mathcal{X}_i for each variable i to be larger than just $\{0, 1\}$. We minimize the cost of the labeling $f : \mathcal{X} \rightarrow \mathbb{R}$ defined by

$$f(\mathbf{x}) = \sum_i f_i(x_i) + \sum_{C \in \mathcal{C}} f_C(\mathbf{x}_C). \quad (7.1)$$

f_C is a function of just the variables x_i with $i \in C$ (a subvector of \mathbf{x} which we denote \mathbf{x}_C). We assume, without loss of generality, that $f_i, f_C \geq 0$ (by reparametrizing them using Lemma 19 of Section 2.1). Special cases include first-order MRFs where $|C| = 2$, and binary MRFs where $|\mathcal{X}_i| = 2$; we are interested in the general

case of higher-order multi-label MRFs, where we restrict neither $|C|$ nor $|\mathcal{X}_i|$.

For multi-label MRFs, the most popular graph-cuts algorithms are based on alpha-expansion. Recall that alpha-expansion solves a series of binary problems, where each variable i can either keep its current label x_i , or switch to a fixed label α . Alpha-expansion cycles through all $\alpha \in \mathcal{X}_i$ until no variable changes in an entire loop through all α .

For certain classes of pairwise function $f_{i,j}$, alpha-expansion has provable approximation guarantees. If the $f_{i,j}$ are all Potts terms, then alpha-expansion is a 2 approximation. When the $f_{i,j}$ are all the same, and form a metric, then the work of [47] showed that the approximation ratio is $2 \frac{f^{\max}}{f^{\min}}$ where f^{\max} is the maximum value of $f_{i,j}$ and f^{\min} is the minimum nonzero value of $f_{i,j}$.

The connection between graph cuts and primal-dual techniques was established by [57] who showed that α -expansion could be interpreted as simultaneously optimizing primal and dual solutions. [57] proposed several primal-dual algorithms that generalized α -expansion and provided both theoretical and practical advantages. These methods apply to much more general energy functions and extend the approximation bounds of [47]. Empirically, keeping track of the dual variables allows a number of implementation speedups compared to α -expansion, resulting in the very efficient algorithm FastPD [59].

7.1.1 Summary of Our Method

In this chapter, we provide an generalization of the primal-dual algorithm PD3 of [57] that can efficiently minimize an arbitrary higher-order multilabel energy

function. Briefly: PD3 relies on the max-flow / min-cut algorithm; the flow values update the dual variables and the min-cut updates the primal variables. Our method instead uses the SoS flow of Chapter 5 which can exactly minimize the class of Sum-of-Submodular functions, with a corresponding SoS max-flow.

Primal-dual methods rely on the optimality conditions for linear programming, in particular the complementary slackness conditions of Section 2.8. These conditions relate the slack in a constraint of the primal problem with the non-zeros of dual variables, and give necessary conditions for a pair of primal and dual solutions to be optimal.

Our algorithm begins with the Local Marginal Polytope (LMP) relaxation of Section 2.4. Recall that the LMP has two kinds of constraints, corresponding to the unary terms and clique-based terms in equation (7.1). We refer to the respective complementary slackness conditions as unary and clique slackness conditions. We will keep track of a primal solution \mathbf{x} and (not necessarily feasible) dual solution λ . We will ensure that \mathbf{x}, λ always satisfy the clique slackness conditions, and at each step of the algorithm, we will try to move \mathbf{x} to be closer to satisfying unary slackness. The algorithm converges to a solution where both slackness conditions hold, but we generally lose feasibility. However, there exists some ρ such that λ/ρ is dual-feasible. This gives us a ρ -approximation algorithm for a class of functions we call weakly associative.

We review the related work in Section 7.2. Our algorithm is presented in Section 7.3, and we conclude with an experimental evaluation in Section 8.3.

7.2 Related Work

7.2.1 Graph Cut Methods and Higher-Order MRFs

The most popular graph cut methods for multilabel first-order MRFs rely on move-making techniques. Those methods, which notably include α -expansion [12] and fusion moves [66], reduce the multilabel problem to a series of binary subproblems which are then solved by max-flow [8, 52]. In α -expansion [12], the binary problem involves each pixel deciding whether to keep its current label or adopt a particular new label α . The expansion move algorithm also provides a guaranteed approximation bound.

[57, 58] proposed a primal-dual framework that generalizes α -expansion. They interpreted this algorithm as optimizing the primal and dual problem of the LP-relaxation of the MRF energy function simultaneously. In addition, the general primal-dual algorithm overcomes the most important limitation of the α -expansion algorithm, which is the requirement that the pairwise energy must be a metric [12]. The same approximation ratio still holds for a much broader class of energy functions. Furthermore, by tracking the dual variables to speed up the optimization, it can be 3-9 times faster in practice [59].

7.2.2 Linear Programming and Duality for MRFs

Much of the theory of MRF optimization algorithms revolves around a specific linear programming relaxation of (7.1) known as the Local Marginal Polytope formulation [86], which was extended to higher-order MRFs in [99]. Every lin-

ear program (LP) has a corresponding dual, and the dual program has resulted in efficient algorithms such as [56, 57, 59]. We derived the dual program for the Local Marginal Polytope in Section 2.9.

Recall that the dual program has variables for each clique C , $i \in C$ and label x_i , denoted $\lambda_{C,i}(x_i)$; and is given by

$$\max_{\lambda} \sum_i \min_{x_i} h_i(x_i) \quad (7.2a)$$

$$h_i(x_i) = f_i(x_i) + \sum_C \lambda_{C,i}(x_i) \quad \forall i \quad (7.2b)$$

$$\sum_{i \in C} \lambda_{C,i}(x_i) \leq f_C(\mathbf{x}_C) \quad \forall C, \mathbf{x}_C \quad (7.2c)$$

We can informally think of the dual variable $\lambda_{C,i}(x_i)$ as taking part of the cost $f_C(\mathbf{x}_C)$, and redistributing it to the unary terms. Following [57], the functions $h_i(x_i)$ will be called the “height” of label x_i at variable i , and semantically can be thought of as the original cost $f_i(x_i)$, plus any redistribution $\lambda_{C,i}$ from the cliques to the unary terms at i . The dual is always a lower bound on the value $f(\mathbf{x})$ of any labeling.

7.2.3 Sum-of-Submodular Flow

We will summarize the most important features of SoS flow from Chapter 5. We have a set of vertices \mathcal{V} plus the source s and sink t , and arcs (s, i) and (i, t) for each $i \in \mathcal{V}$. We are also given a *sum-of-submodular function*:

$$g(S) = \sum_{C \in \mathcal{C}} g_C(S \cap C) + \sum_{i \in S} c_{i,t} + \sum_{i \notin S} c_{s,i} \quad (7.3)$$

where $C \in \mathcal{C}$ are called cliques in \mathcal{V} , and each g_C is a *submodular function*, called a clique function, with

$$g_C(\emptyset) = g_C(C) = \min_S g_C(S \cap C) = 0. \quad (7.4)$$

Intuitively, the difference between max flow and sum-of-submodular flow is that in addition to capacity and conservation constraints, we will also require that the flow out of any set S is at most $g_C(S \cap C)$. To be precise, a sum-of-submodular flow has flow values $\phi_{s,i}$ and $\phi_{i,t}$ on the source and sink edges, as well as flow values $\phi_{C,i}$ for each clique C and $i \in C$. Then, a maximum sum-of-submodular flow is a solution to the following LP:

$$\max_{\phi} \sum_i \phi_{s,i} \quad (7.5a)$$

$$\text{s.t. } \phi_{s,i} \leq c_{s,i}, \quad \phi_{i,t} \leq c_{i,t} \quad \forall i \quad (7.5b)$$

$$\phi_{s,i} - \phi_{i,t} - \sum_{C \ni i} \phi_{C,i} = 0 \quad \forall i \quad (7.5c)$$

$$\sum_{i \in S} \phi_{C,i} \leq g_C(S) \quad \forall C, S \subseteq C \quad (7.5d)$$

Here, (7.5b) are the capacity constraints for source and sink edges, with capacities given by the unary terms $c_{s,i}, c_{i,t}$, (7.5c) are the flow-conservation constraints at i and (7.5d) are the additional constraints that the ϕ_C in a set S are at most $g_C(S)$. [53] shows that this LP can be solved by a generalized flow algorithm.

Finally, we have a sum-of-submodular version of the min-cut max-flow theorem, originally from [53], and described in Section 5.1.3. If ϕ maximizes (7.5), and S minimizes (7.3), then the objective value (7.5a) of ϕ is equal to $g(S)$. Furthermore, the notion of saturated edges extends to the clique function: (1) if

```

Initialize  $\mathbf{x}$  arbitrarily.
Initialize  $\lambda_{C,i}(x_i) = \frac{1}{|C|} f_C(\mathbf{x}_C)$ , and  $\lambda_{C,i}(a) = 0$  for  $a \neq x_i$ .
while unary slackness conditions are not satisfied do
     $\mathbf{y} \leftarrow$  result of proposal generator
    PRE-EDIT-DUALS( $\mathbf{x}, \mathbf{y}, \lambda$ )
     $\mathbf{x}', \lambda' \leftarrow$  UPDATE-DUALS-PRIMALS( $\mathbf{x}, \mathbf{y}, \lambda$ )
    POST-EDIT-DUALS( $\mathbf{x}', \lambda'$ )
end while
return  $\mathbf{x}$ 

```

Algorithm 1: Our SoSPD algorithm.

$i \in S$ then $\phi_{i,t} = c_{i,t}$ (2) if $i \notin S$ then $\phi_{s,i} = c_{s,i}$, and most importantly (3) for every clique C , $g_C(S \cap C) = \sum_{i \in S} \phi_{C,i}$.

7.3 The SoS Primal Dual Algorithm

Our algorithm, which we will call SoSPD, is designed around ensuring that two main conditions are satisfied regarding the primal and dual solutions. These conditions give us our approximation bound, as well as help design the rest of the algorithm. The conditions are complementary slackness conditions (Section 2.8), in which the inequalities in the dual that correspond to a particular primal solution are actually satisfied with equality.

Definition 79. *Given a labeling \mathbf{x} and dual solution λ , we say that \mathbf{x}, λ satisfy the clique slackness conditions if the constraints in (7.2c) corresponding to \mathbf{x}_C are satisfied with equality. That is, we have*

$$\sum_{i \in C} \lambda_{C,i}(x_i) = f_C(\mathbf{x}_C) \quad \forall C \quad (7.6)$$

Proposition 80. *If \mathbf{x}, λ satisfy the clique slackness conditions, then $f(\mathbf{x}) = \sum_i h_i(x_i)$.*

Proof. Remembering our redistribution argument, this means we have exactly partitioned $f_C(\mathbf{x}_C)$ among the λ , so the sum of the heights is the original cost $f(\mathbf{x})$.

That is,

$$\begin{aligned} \sum_i h_i(x_i) &= \sum_i \left(f_i(x_i) + \sum_C \lambda_{C,i}(x_i) \right) \\ &= \left(\sum_i f_i(x_i) \right) + \left(\sum_C \sum_i \lambda_{C,i}(x_i) \right) \\ &= \sum_i f_i(x_i) + \sum_C f_C(\mathbf{x}_C) = f(\mathbf{x}) \quad \square \end{aligned} \tag{7.7}$$

Definition 81. \mathbf{x}, λ satisfy the unary slackness conditions if for each i we have $h_i(x_i) = \min_a h_i(a)$.

Corollary 82. If \mathbf{x}, λ satisfy both the clique and unary slackness conditions, and λ is feasible, then \mathbf{x} minimizes f .

Proof. From Proposition 80, the sum of heights $\sum_i h_i(x_i)$ is equal to $f(\mathbf{x})$, and by the definition of unary slackness, the sum of heights is also equal to the dual objective, the lower-bound on all possible values $f(\mathbf{x})$. \square

Since our original problem is NP-hard we can't expect both slackness conditions to hold for a feasible dual λ and integral primal \mathbf{x} (for any solution we can find in polynomial time). We instead apply a technique called dual scaling [57], in which we allow our duals to become slightly infeasible, but in a way that they can be multiplied by a scalar to become feasible. More specifically, the structure of (7.2) always allows us to scale down λ by $\frac{1}{\rho}$ for some $\rho \geq 1$ to get a feasible solution. This gives us approximate optimality.

Lemma 83. If \mathbf{x} and λ satisfy the unary and clique slackness conditions, and λ/ρ is dual feasible, then $f(\mathbf{x}) \leq \rho f(\mathbf{x}^*)$, where \mathbf{x}^* is the true optimum.

Proof. Since \mathbf{x}, λ satisfy both slackness conditions, we know that $f(\mathbf{x}) = \sum_i \min_{a_i} h_i(a_i)$, hence

$$\begin{aligned} f(\mathbf{x}) &= \rho \sum_i \min_{a_i} \frac{1}{\rho} [f_i(a_i) + \sum_c \lambda_{c,i}(a_i)] \\ &\leq \rho \sum_i \min_{a_i} [f_i(a_i) + \sum_c \frac{1}{\rho} \lambda_{c,i}(a_i)] \leq \rho f(\mathbf{x}^*) \end{aligned}$$

where the first inequality is because $f_i \geq 0$, and the second from λ/ρ being dual-feasible. \square

Lemma 83 gives the basic motivation behind our algorithm. Between iterations, \mathbf{x}, λ will always satisfy the clique slackness conditions, and the goal of each iteration is to change \mathbf{x} to move to lower height labels. At the end of the algorithm, all the x_i will be the lowest height labels for each i , and the unary slackness conditions are satisfied. Then, we'll prove that there exists some ρ such that λ/ρ is dual-feasible, and hence we have a ρ -approximation algorithm.

The difficult step in this algorithm is that when we change the labeling \mathbf{x} to decrease the height, we must still maintain the clique slackness conditions. We cannot simply set each x_i to the lowest height label, lest the clique slackness conditions cease to hold. Instead we simultaneously pick a set of labels to change, and adjust the dual variables such that the new clique slackness conditions are tight. For the higher order case, we can show that sum-of-submodular flow is exactly the tool we need to ensure the clique slackness conditions still hold when changing labels.

At a high-level, the algorithm works as follows. At each iteration, much like the α -expansion or fusion move algorithms, we have a current labeling \mathbf{x} and a proposed labeling \mathbf{y} . We use sum-of-submodular flow to pick a set S of variables that switch labels, and the max-flow min-cut theorem for sum-of-submodular

flow will ensure that the new variables \mathbf{x}', λ' also satisfy the clique slackness conditions.

Our SoSPD technique is summarized in Algorithm 1, and each iteration has 3 subroutines. The main work of the algorithm occurs in UPDATE-DUALS-PRIMALS, which sets up the sum-of-submodular flow problem, and picks a set of variables to swap. We will describe this subroutine first, in Section 7.3.1, making some assumptions about \mathbf{x}, λ which may not hold in general. Then, it is the job of the other two subroutines, PRE-EDIT-DUALS and POST-EDIT-DUALS (Sections 7.3.2 and 7.3.3) to make sure these assumptions do hold, and that therefore the algorithm functions correctly.

7.3.1 Update-Duals-Primals

To begin with, we need notation for fusion moves [66]. If we have current and proposed labelings \mathbf{x} and \mathbf{y} , and S is the set of variables that change label, we'll denote the fused labeling by $\mathbf{x}' = \mathbf{x}[S \leftarrow \mathbf{y}]$, which has $x'_i = y_i$ if $i \in S$, and $x'_i = x_i$ if $i \notin S$.

Given our current state \mathbf{x}, λ , we're going to construct a sum-of-submodular flow network. The values $\phi_{C,i}$ will be the amount we add or subtract from $\lambda_{C,i}(\mathbf{y}_i)$, and the source-sink flow $\phi_{s,i}, \phi_{i,t}$ will give the change in height of $h_i(\mathbf{y}_i)$. We will only ever adjust the dual variables $\lambda_{C,i}(\mathbf{y}_i)$ corresponding to the proposed labeling \mathbf{y} .¹

The easy part is defining the source-sink capacities. If $h_i(\mathbf{y}_i) < h_i(\mathbf{x}_i)$ then we

¹Note that if $x_i = y_i$, we do not change $\lambda_{C,i}(x_i)$. We could accomplish this by simply removing such i from the flow network. However such vertices i will have, by construction, no outgoing capacity in the network, so $\phi_{C,i}$ must always be 0.

can raise the height of label y_i by the difference, and still prefer to switch labels. Similarly, if $h_i(y_i) > h_i(x_i)$, we can lower the height of y_i by the difference without creating a new label we'd prefer to swap to. We define source-sink capacities by $c_{s,i} = h_i(x_i) - h_i(y_i)$, $c_{i,t} = 0$ if $h_i(y_i) > h_i(x_i)$, and $c_{s,i} = 0$, $c_{i,t} = h_i(y_i) - h_i(x_i)$ otherwise.

In addition to decreasing the heights of the variables, our other main concern is making sure that the clique slackness conditions continue to hold. Consider an individual clique C for now, and let us examine what our labeling \mathbf{x}'_C could look like after a fusion step. The possible labelings are $\mathbf{x}_C[S \leftarrow \mathbf{y}_C]$ for each subset S of C . We want to make sure that after the swap, $\sum_i \lambda_{C,i}(x'_i) = f_C(\mathbf{x}'_C)$, so define a function g_C equal to the difference:

$$g_C(S) := f_C(\mathbf{x}_C[S \leftarrow \mathbf{y}_C]) - \sum_{i \in S} \lambda_{C,i}(y_i) - \sum_{i \notin S} \lambda_{C,i}(x_i) \quad (7.8)$$

For now, we'll assume that (1) g_C is a submodular function and (2) $g_C(\emptyset) = g_C(C) = 0$, $g_C(S) \geq 0$. These assumptions will end up being enforced by PRE-EDIT-DUALS, which we describe below.

Under these assumptions the capacities c and functions g_C define a sum-of-submodular flow network, so we can find a flow ϕ and cut S such that $g_C(S \cap C) = \sum_{i \in S} \phi_{C,i}$ (by the sum-of-submodular version of the max-flow min-cut theorem [53], paraphrased at the end of Section 7.2.3). Then, we set $\mathbf{x}' = \mathbf{x}[S \leftarrow \mathbf{y}]$, and $\lambda'_{C,i}(y_i) = \lambda_{C,i}(y_i) + \phi_{C,i}$. By definition of g_C , we have

$$\begin{aligned} f_C(\mathbf{x}'_C) &= g_C(S \cap C) + \sum_{i \in S} \lambda_{C,i}(y_i) + \sum_{i \notin S} \lambda_{C,i}(x_i) \\ &= \sum_{i \in S} [\lambda_{C,i}(y_i) + \phi_{C,i}] + \sum_{i \notin S} \lambda_{C,i}(x_i) = \sum_i \lambda'_{C,i}(x'_i). \end{aligned}$$

Therefore, the primal and dual solutions satisfy the clique slackness condi-

tions, and our source-sink capacities were chosen so that $h'_i(x'_i) \leq h_i(x_i)$.

Finally, note that unless every edge out of s gets saturated (and hence $S = \emptyset$) then at least one height has strictly decreased.

7.3.2 Pre-Edit-Duals

The job of PRE-EDIT-DUALS is to ensure that the assumptions we made in UPDATE-DUALS-PRIMALS are actually true. Namely, we need (1) the function g_C must be submodular and (2) $g_C(\emptyset) = g_C(C) = 0$ and $g_C(S) \geq 0$.

For (1), first note that if $f_C(\mathbf{x}_C[S \leftarrow \mathbf{y}_C])$ is a submodular function of S , then so is g_C , since a submodular function plus a linear function is still submodular. Such functions were called *expansion-submodular* in [19]. To handle general energy functions, we need an approach for the case where the fusion move is not submodular.

We take a similar approach to the PD3 variant PD3_a [57], which finds an overestimate of the original energy function. For pairwise energies finding a submodular overestimate simply consists of truncating negative capacities to 0. In our case, we must find a submodular upper bound, $\tilde{f}_C(S)$, such that $\tilde{f}_C(S) \geq f_C(\mathbf{x}_C[S \leftarrow \mathbf{y}_C])$. Our only other requirements are that $\tilde{f}_C(\emptyset) = f_C(\mathbf{x}_C)$, $\tilde{f}_C(C) = f_C(\mathbf{y}_C)$, and that $\tilde{f}(\{i\}) \leq \max_{\mathbf{x}_C} f_C(\mathbf{x}_C)$ for $i \in C$. We will use the methods of Chapter 6 for finding submodular upper bounds of functions. We consider each of the choices presented there in the experiments.

Having computed \tilde{f}_C , we then substitute it for f_C , just for this iteration. To simplify the notation we will write $\tilde{f}_C(\mathbf{x}'_C)$ to mean $\tilde{f}_C(S)$ wherever $\mathbf{x}'_C = \mathbf{x}_C[S \leftarrow$

$\mathbf{y}_C]$.

To establish assumption (2), we make use of Edmonds algorithm [16], described by Lemma 35 from Section 2.3.3. This states that for any submodular function g with $g(\emptyset) = 0$, there is a vector ψ such that $g(S) + \psi(S) \geq 0$ and $g(C) = -\psi(C)$ (where we are using the standard notation $\psi(S) := \sum_{i \in S} \psi_i$). In fact, the vector defined by $\psi_i = g(\{1, \dots, i-1\}) - g(\{1, \dots, i\})$ will suffice.

To ensure (2) holds, we start with $g_C(S)$ defined as in (7.8). Note that we have $g_C(\emptyset) = f_C(\mathbf{x}_C) - \sum_{i \in C} \lambda_{C,i}(x_i)$, which by the clique slackness condition, we know is 0. We can therefore compute a ψ as just described, and update $\lambda_{C,i}(y_i) \leftarrow \lambda_{C,i}(y_i) - \psi_i$. Since $g_C(S) + \psi(S) \geq 0$ and $g_C(C) + \psi(C) = 0$, when we update $g_C \leftarrow g_C + \psi$ with the new values of λ , we satisfy $g_C(S) \geq 0$ and $g_C(C) = 0$.

7.3.3 Post-Edit-Duals

Having run UPDATE-DUALS-PRIMALS, we know that $\tilde{f}_C(\mathbf{x}'_C) = \sum_i \lambda'_{C,i}(\mathbf{x}'_i)$. However, from PRE-EDIT-DUALS, \tilde{f} might be an overestimate of f .

The subroutine POST-EDIT-DUALS enforces the clique slackness conditions, by setting $\lambda'_{C,i}(y'_i) = \frac{1}{|C|} f_C(\mathbf{x}'_C)$ for each clique C . Note that if \tilde{f} is an overestimate, this can only ever decrease the sum of heights $h_i(x'_i)$ (since we first average, and then subtract the overestimate from λ).

One final property of POST-EDIT-DUALS: since $f_C(\mathbf{x}'_C) \geq 0$, we always know that $\lambda_{C,i}(x'_i) \geq 0$. We will use this in the proof of approximation ratio, momentarily.

7.3.4 Proof of Convergence

Much like the pairwise algorithms α -expansion and fusion move, we have monotonically decreasing energy.

Lemma 84. *The objective value $f(\mathbf{x})$ is non-increasing.*

Proof. First, recall that \mathbf{x}, λ satisfy the clique slackness conditions, so $f(\mathbf{x}) = \sum_i h_i(x_i)$. We also know that PRE-EDIT-DUALS doesn't change any of the heights $h_i(x_i)$, UPDATE-PRIMALS-DUALS can only decrease $h_i(x_i)$ (by definition of the source-sink capacities) and POST-EDIT-DUALS also doesn't increase the sum of heights. \square

The convergence of our method is not guaranteed for arbitrarily bad fusion moves (for instance, we could have a bad proposal generator which always suggests labels which have greater height than x_i). For α -expansion proposals, however, convergence is guaranteed.

Proposition 85. *With the proposal $\mathbf{y}_i = \alpha$ for each i , at the end of the iteration either $f(\mathbf{x}') < f(\mathbf{x})$ or $h_i(x_i) \leq h_i(\alpha)$ for all i .*

Proof. From the discussion of UPDATE-DUALS-PRIMALS, one of two things happens: (1) the height of at least one variable is strictly decreased, or (2) the minimum cut is $S = \emptyset$. If (1), then neither of the other subroutines increases the sum of heights, so by Proposition 80 we have $f(\mathbf{x}') < f(\mathbf{x})$. If (2) then all edges out of s are saturated, so UPDATE-DUALS-PRIMALS increased $h_i(\alpha)$ to be at least $h_i(x_i)$. Furthermore, $\mathbf{x}' = \mathbf{x}$ and so neither PRE-EDIT-DUALS nor POST-EDIT-DUALS changes any of the $\lambda_{C,i}(x_i)$, and therefore $h_i(x_i) \leq h_i(\alpha)$ holds at the end of the iteration. \square

Lemma 86. *If after running through iterations of α -expansion for every label α , $f(\mathbf{x})$ does not strictly decrease, then the unary slackness conditions must hold, and the algorithm terminates.*

Proof. Since every α -expansion iteration didn't change the objective $f(\mathbf{x})$, by Proposition 85 each such iteration ensures that $h_i(\alpha) \geq h_i(x_i)$. Also note that a β -expansion for $\beta \neq \alpha$ doesn't change any of the $h_i(\alpha)$. Therefore, the x_i are all minimum height labels, and the unary slackness conditions are satisfied. \square

Overall, with integer costs, the objective decreases by at least 1 each outer-iteration and therefore eventually halts. The running time of each iteration is dominated by the SoS flow computation — we use SoS-IBFS [19] which has runtime $O(|\mathcal{V}|^2|C|2^k)$, where $k = \max |C|$. It is difficult to provide a non-trivial bound on the number of α -expansion iterations, but in practice we always observe convergence after 4 passes through the label set. Note that this is exponential in the clique size, since we represent submodular functions as tables of 2^k values. However, this is also true of other state of the art methods for higher-order MRFs such as [18, 40, 55].

7.3.5 Approximation Bounds

Let $f^{\max} = \max_C f_C(\mathbf{x}_C)$, $f^{\min} = \min_C f_C(\mathbf{x}_C)$, where the max and min are over all cliques C and all non-constant labelings \mathbf{x}_C (meaning there is no a with $x_i = a$ for all $i \in C$). There is a natural class of MRFs where $f^{\min} > 0$ (i.e. all non-constant labelings have positive costs), and where constant labelings have zero cost. We call such MRFs *weakly associative*; they encourage all variables in a clique to have

the same label, but are otherwise unrestricted on non-constant labelings. This generalizes what [57] calls non-metric energies.

Our approximation ratio will be $\rho = k \frac{f^{\max}}{f^{\min}}$. Note that ρ is finite only for a weakly associative MRF. This generalizes the approximation ratio for PD3, which is $2 \frac{f^{\max}}{f^{\min}}$.

Theorem 87. *SoSPD with α -expansion for a weakly associative MRF f is a ρ -approximation algorithm, i.e., the primal solution \mathbf{x} at the end will have $f(\mathbf{x}) \leq \rho f(\mathbf{x}^*)$.*

Proof. The first task is to show that λ doesn't get too big. In particular, after any iteration, $\lambda_{C,i}(a_i) \leq f^{\max}$ for all a_i . Note that after UPDATE-DUALS-PRIMALS, we have

$$\phi_{C,i} \leq g_C(\{i\}) := \tilde{f}_C(\{i\}) - \sum_{j \neq i} \lambda_{C,i}(x_i) - \lambda_{C,i}(y_i)$$

Since we constructed \tilde{f} to have $\tilde{f}(\{i\}) \leq f^{\max}$ and POST-EDIT-DUALS from the previous iteration makes sure $\lambda_{C,i}(x_i) \geq 0$, we get $\lambda'_{C,i}(y_i) = \lambda_{C,i}(y_i) + \phi_{C,i} \leq f_C^{\max}$. If POST-EDIT-DUALS in the present iteration changes $\lambda_{C,i}(y_i)$, it sets it to $\frac{1}{|C|} f_C(\mathbf{x}') \leq f_C^{\max}$. Therefore, $\lambda'_{C,i}(y_i) \leq f_C^{\max}$, and we don't change $\lambda_{C,i}(a_i)$ for any $a_i \neq y_i$ in this iteration, so inductively, at the end of the algorithm $\lambda_{C,i}(a_i) \leq f_C^{\max}$ for all labels a_i .

For feasibility, we need to show that (7.2c) holds for each clique C and labeling \mathbf{x}_C . For non-constant \mathbf{x}_C we have

$$\sum_i \frac{1}{\rho} \lambda_{C,i}(x_i) \leq \frac{|C| f_C^{\max}}{\rho} \leq f_C^{\min} \leq f_C(\mathbf{x}_C)$$

For constant labeling $\mathbf{x}_C = \alpha$, note that in the last α -expansion, PRE-EDIT-DUALS enforces that $\tilde{f}_C(C) - \sum_i \lambda_{C,i}(\alpha) = g_C(C) = 0$, and neither of the other subroutines violate this. Therefore $\sum_i \frac{1}{\rho} \lambda_{C,i}(\alpha) = \frac{1}{\rho} \tilde{f}_C(C) = \frac{1}{\rho} f_C(\alpha) = 0$, where the second

equality is because we constructed \widetilde{f}_C with $\widetilde{f}_C(C) = \widetilde{f}(\mathbf{y}_C)$, and the last is since f is weakly associative.

Finally, Lemma 83 says that at convergence, $f(\mathbf{x})$ is no more than $\rho f(\mathbf{x}^*)$. \square

CHAPTER 8

EXPERIMENTAL EVALUATION OF THE SOSPD ALGORITHM

The last two chapters presented an algorithm for optimizing general non-submodular higher-order MRFs, based off linear programming relaxations to the local marginal polytope, and using submodular flow and submodular upper bounds to solve the binary subproblems in each fusion or expansion move. In this chapter, we will give the experimental results for these algorithms, showing that they are also empirically faster than existing state of the art algorithms.

For our benchmarks, we choose the two exemplar higher-order problems described in the introduction: the Field of Experts model from Section 1.5.2 and the curvature regularizing stereo model of Section 1.5.3. We will describe in detail the datasets and models used for the experiments in Section 8.1.

There are two main questions we want to answer with these experiments. In Section 8.2, we explore which of the several proposed submodular upper bounds is the most effective for optimization in typical computer vision input problems. Then, in Section 8.3 we demonstrate the speedup of the primal-dual SoSPD algorithm over existing algorithms for multilabel higher-order inference.

8.1 Benchmarks and Datasets

For our experiments, we are interested in benchmarks which represent typical higher-order vision inputs, and which are difficult to optimize.

8.1.1 Field of Experts Denoising

The first benchmark, Fields of Experts denoising, we have already seen in the experiments for the reduction method of Chapter 4. This benchmark is based off the model of [77], and has been used in many higher-order optimization papers, including [40, 18, 43, 22]. The dataset consists of 100 grayscale images from the Berkeley Segmentation Database [70], to which independent Gaussian noise has been added to each pixel.

Recall that the Field of Experts model has 255 labels for each pixel x_i , with labels corresponding to denoised 8-bit intensity values. The unary terms are a L_2 data-cost

$$\sum_i \frac{1}{2\sigma^2} \|x_i - y_i\|^2 \quad (8.1)$$

where y_i is the observed, noisy image and σ is an estimate of the gaussian noise added to each pixel.

The Field of Experts prior is applied to each local patch of the image. We use each (overlapping) 2×2 patch, for cliques of size 4. The clique functions are given by

$$f_C(\mathbf{x}_C) = \sum_{i=1}^k \alpha_i \log(1 + J_i^T \mathbf{x}_C). \quad (8.2)$$

where the J_i are learned linear filters passed through a nonlinear activation function $\log(1 + \cdot)$, and α_i are learned mixture components between these activations. The exact coefficients were trained by maximum-likelihood estimation on a training set. To allow reproducibility of results, we obtained the specific energy functions to be minimized from the OpenGM benchmark [45].¹

¹Available at <http://hci.iwr.uni-heidelberg.de/opengm2/>.

8.1.2 Curvature Regularizing Stereo Reconstruction

The second benchmark is based on the second-order stereo model of [102], as described in Section 1.5.3. The stereo reconstruction algorithm of [102] encourages the disparity map to be piecewise smooth using a 2nd order prior, composed of all 1×3 and 3×1 patches in the image, which each penalizing a robust function of the curvature of the disparity map.

A number of optimization methods are proposed in [102], which are composed to get the final result. The most important step consists of pre-generating a set of 14 piecewise-planar proposed disparity maps, and then using these as proposals to the fusion-move algorithm to improve the current disparity until convergence. This method is called SEGPLN in [102]. We use the truncated-quadratic costs which penalizes the deviation of each 3×1 patch from a plane, via the robust L_2 cost:

$$f_{i,j,k}(x_i, x_j, x_k) = \min \left\{ \left| \frac{\Delta}{x_i} - 2\frac{\Delta}{x_j} + \frac{\Delta}{x_k} \right|^2, \tau \right\}. \quad (8.3)$$

To give a fair benchmark comparison, we use the simplified model used in [55] and later adapted by [22], which omits the binary occlusion labels for each pixel. Because the stereo-reconstruction problem is doing inference in a continuous domain (of all possible disparity values, as real numbers) this model also discretizes the problem by giving each pixel 14 discrete labels, one for each pre-generated proposal. Experimentally, the fusion-move part of [102] (in which the algorithm repeatedly proposes and fuses these 14 proposals) is the bulk of the time spent by the algorithm, as well as the most important for energy reduction. So, good performance of discrete optimizers for this step can dramatically improve the performance of the whole algorithm.

Note that this discretization still allows estimating sub-pixel disparities (since the proposals can take any floating point value) using only 14 labels.

Data was obtained by running the code² for [102] and recording the proposed fusion moves and corresponding unary terms. The dataset consists of 3 stereo pairs, “cones”, “teddy” and “venus” from the Middlebury Stereo Dataset [82, 83] obtained from <http://vision.middlebury.edu/stereo/data/>.

8.2 Comparison of Upper Bound Methods

8.2.1 Experimental Setup

Our first set of experiments tests the effectiveness of the various upper bound methods of Chapter 6. We are primarily interested in the performance of these upper bounds as a subroutine within the SoSPD algorithm, as SoSPD can handle multilabel higher-order problems, including the two benchmarks of Section 8.1. It is possible to apply submodular upper bounds directly to the optimization of non-submodular higher-order binary problems; however, these problems are typically much less interesting from an application perspective as most computer vision problems are multilabel.

In addition to SoSPD, we also test the effectiveness of the upper bounds in a fusion move algorithm [66], by using the submodular upper bound to convert each (possibly non-submodular) higher-order binary fusion move into a

²<http://www.robots.ox.ac.uk/~ojw/software.htm>.

Method	Energy	Time (s)	% Best
SoSPD-QUAD	30712.58	16.049	0.00%
SoSPD-HEUR	30748.19	20.212	0.00%
SoSPD-CARD	31190.61	22.179	0.00%
SoSPD-LP ₁	30706.60	2099.896	10.00%
SoSPD-LP _∞	30704.49	5233.162	90.00%

Table 8.1: Comparison of upper bound methods for Fields of Experts denoising averaged over 10 images. Results for SoSPD-LP₁ and SoSPD-LP_∞ computed with the Gurobi LP solver. Gradient descent proposals were used to generate fusion moves in SoSPD.

Method	Energy	Time (s)	% Best
SoSPD-QUAD	32593.23	15.881	100.00%
SoSPD-CARD	33074.61	21.908	0.00%
SoSPD-HEUR	32629.35	20.032	0.00%

Table 8.2: Comparison of upper bound methods for the full Fields of Experts denoising dataset, averaged over 100 images. Gradient descent proposals used for SoSPD and REDUCTION-FUSION.

Method	Energy	Time (s)	% Best
SoSPD-QUAD	8.958×10^9	104.421	0.00%
SoSPD-HEUR	8.952×10^9	102.311	0.00%
SoSPD-CARD	8.958×10^9	116.521	0.00%
SoSPD-LP ₁	8.953×10^9	115.281	0.00%
SoSPD-LP _∞	8.948×10^9	119.012	100.00%

Table 8.3: Comparison of upper bound methods for the Stereo dataset, averaged across 3 stereo pairs, “cones”, “teddy” and “venus”. Results for SoSPD-LP₁ and SoSPD-LP_∞ were computed using our custom simplex implementation.

submodular one, and then solve the resulting SoS optimization problem using the SoS IBFS algorithm of 5. Note that the fusion-move algorithm can be considered a pure-primal version of the primal-dual algorithm SoSPD, in that both algorithms will take the same sequence of fusion or expansion moves, with the same optimal binary labeling at each step. Thus, when both algorithms use the same upper bound, they will arrive at the same answer, though typically SoSPD is more efficient. We observed in our experiments that for a given lower bound, fusion moves had nearly identical³ final energy to the corresponding SoSPD result, but took a little over twice as long on both the stereo and denoising datasets, so we did not include them on the tables here.

Both Fusion-moves and SoSPD allow a choice of fusion proposals at each iteration. For the stereo example, we simply cycle through the 14 labels, doing an expansion move on each one. For the Fields of Experts experiments, we use the gradient descent proposals of [38], which have been shown to be the most effective fusion proposals for this dataset.

For the implementation, we used the publicly available code from [22] for the implementation of sum-of-submodular flow, as well as the SoSPD algorithm. The L_1 and L_∞ linear programs in (6.3) were solved by the linear programming package Gurobi. Additionally, for size 3 cliques in the stereo benchmark, the linear programs involved are very small (only 6 variables and 6 constraints) so we implemented a version of the simplex method with the constraints hard-coded, which was much faster than the general Gurobi solver. All code is in C++, and will be released under an open source license.⁴

We compare five different submodular upper bounds. We give each an ab-

³With differences largely due to stopping conditions

⁴Available at www.cs.cornell.edu/~afix.



Figure 8.1: Visual results for Fields of Experts denoising with different upper bound methods. Top row, left to right: (a) SOSPD-HEUR. Bottom row: (b) SOSPD-QUAD (c) SOSPD-CARD.

breviation in the tables and figures: the p -norm minimizing upper bounds of Section 6.3, equation (6.3), we'll denote as LP_1 and LP_∞ , the quadratic-based approximation of 6.4.2 is QUAD, the cardinality approximation of 6.4.3 is CARD, the baseline heuristic of [22] (in Section 6.4.1) we'll denote by HEUR. We also use, for example, SOSPD-QUAD for the SoSPD algorithm with the quadratic approximation, and FUSION- LP_1 to denote the fusion move algorithm with the 1-norm upper bound, etc.

8.2.2 Results

Our first experiment tests how close our proposed approximations come to the L_1 and L_∞ minimizing upper bounds. Because of the slow-runtime of solving the LP for the L_1 and L_∞ upper bounds, we ran these on only the first 10 images for the Field of Experts dataset. Results of this experiment are summarized in

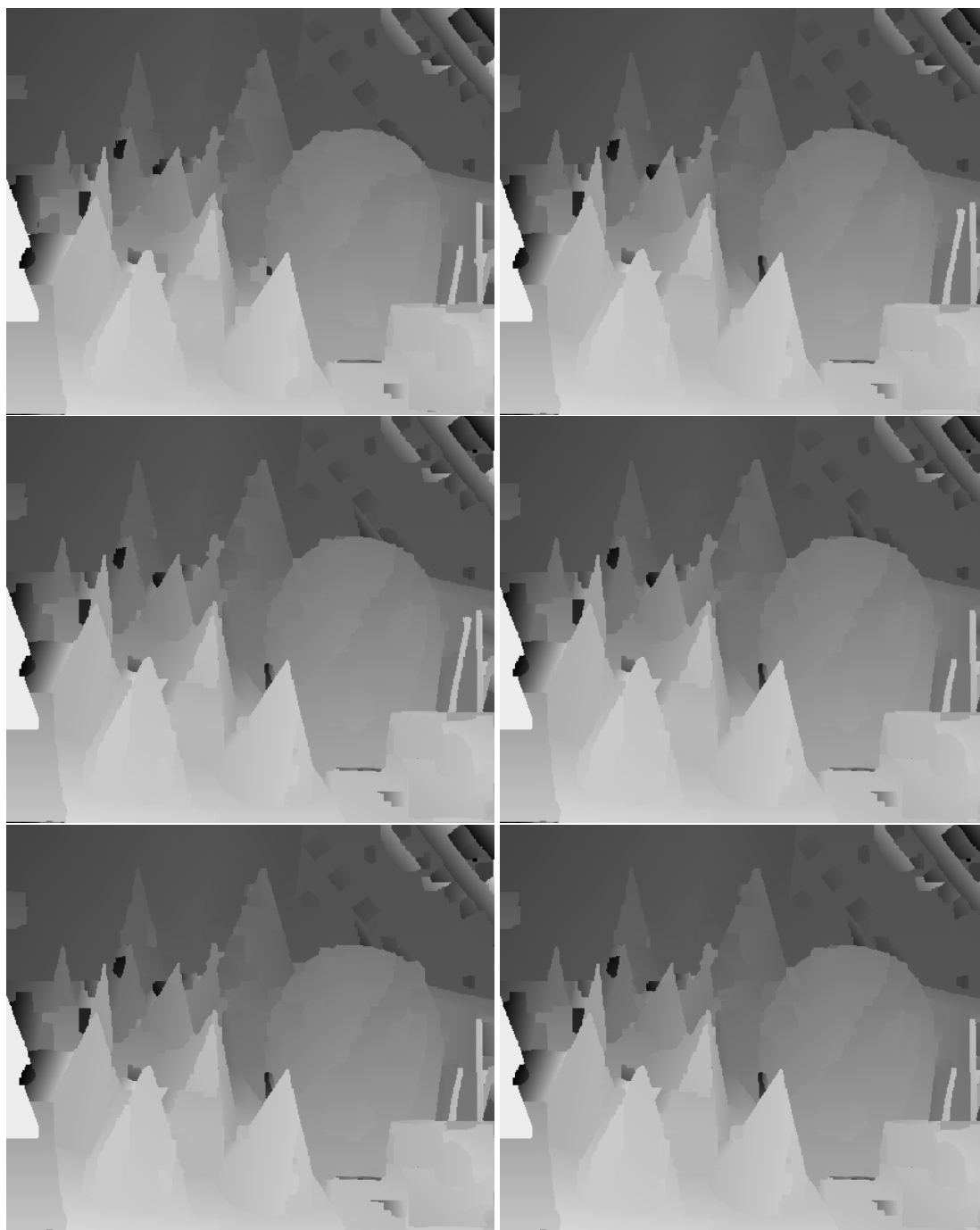


Figure 8.2: Reconstructed depth maps for stereo pair “cones”. Top row, left to right (with % of pixels within ± 1 disparity) (a) REDUCTION-FUSION, 49.0% (b) SoSPD-QUAD, 49.9% Center row (c) SoSPD-CARD, 49.9%. (d) SoSPD-HEUR, 49.7% Bottom row (e) SoSPD-LP₁, 50.0% (f) SoSPD-LP_∞, 49.7%

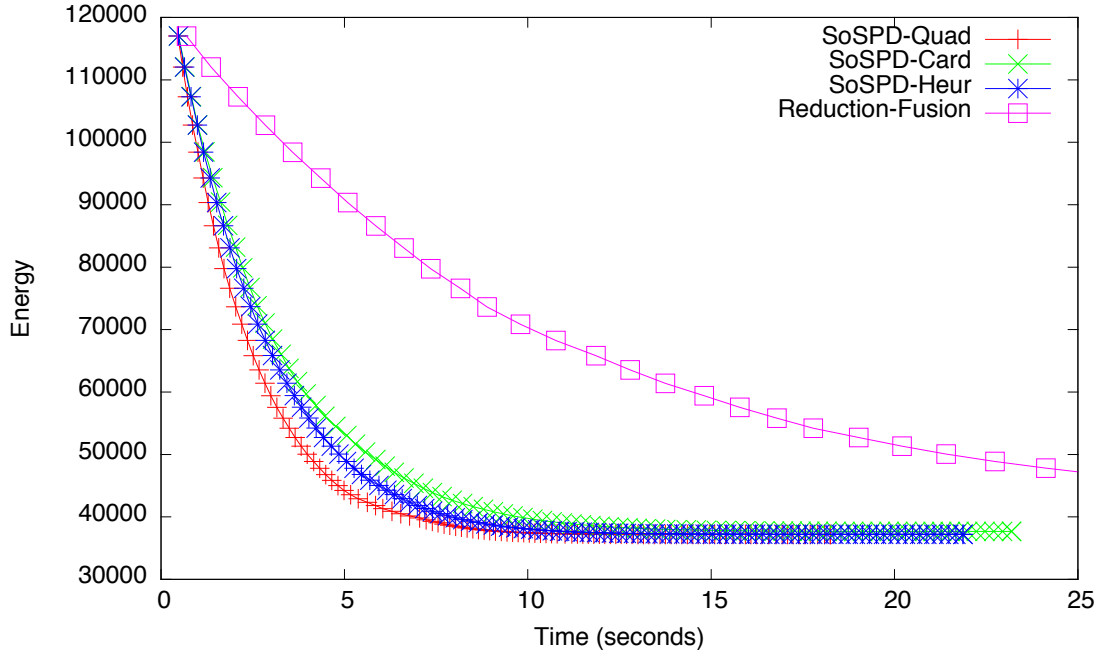


Figure 8.3: Comparison of upper bound methods: Energy over time for the Fields of Experts denoising experiment, using the image in Figure 8.1. Reduction-Fusion, using the reduction method of Chapter 4 provided for comparison.

Table 8.1. Overall, the L_∞ upper bound performed the best in the majority (90%) of instances, while the L_1 upper bound had only slightly higher energy. The norm-minimizing upper bounds together perform better than all other methods, indicating that these norms are a good measure to minimize for picking good upper bounds. Additionally, the pairwise approximation SoSPD-Quad was very close to the L_∞ result, with the energy gap between them less than $1/4$ the gap between SoSPD-Quad and the next competitor, SoSPD-Heur. This suggests that the proposed upper bound approximations can come close to the Linear Programming solution, while being more than 100 times faster.

Next, we run all non-LP methods on the full denoising dataset, with results in Table 8.2. Notably, the pairwise approximation has both the best energy for

every image in the dataset, as well as being the fastest overall.

For the stereo example, results for the 3 stereo pairs are summarized in Table 8.3. Since the cliques were of size 3, we were able to use the custom simplex method mentioned above for the L_1 and L_∞ upper bounds. The L_∞ upper bound achieved the best energy for all 3 images, while taking only 16% more time (and nearly 4x faster than the non upper bound method, REDUCTION-FUSION).

Across both datasets, we find that, the linear programming based upper bounds have the best energy performance, particularly the ∞ -norm upper bound. However, for cliques of size four or more, computing the linear programming solution becomes expensive. Thus, the quadratic-based approximation is also promising, as it is faster than the norm-based methods for both datasets, while still having very similar energy optimization performance. As expected, the methods all achieve very similar final energies, and correspondingly the visual results for all upper bound algorithms are nearly indistinguishable, as seen in Figure 8.1 and 8.2.

8.3 Evaluation of SoSPD

Now that we have identified the best upper bound algorithms for different problems, we now want to compare the performance of SoSPD against existing state-of-the-art algorithms for higher-order multilabel inference. In the following experiments, we use the custom-simplex implementation of SoSPD-LP $_\infty$ for cliques of size 3, and SoSPD-QUAD for larger cliques.

For experimental comparisons, the method of [55] does not currently have

“Teddy”	Pixels within ± 1	Final energy	Time
FGBZ-Fusion	83.3%	9.320×10^9	468s
HOCR-Fusion	83.8%	9.298×10^9	210s
GRD-Fusion	84.9%	9.256×10^9	1116s
SoSPD-Fusion	84.8%	9.172×10^9	129s
“Cones”	Pixels within ± 1	Final energy	Time
FGBZ-Fusion	74.9%	1.1765×10^{10}	340s
HOCR-Fusion	74.2%	1.1789×10^{10}	172s
GRD-Fusion	75.2%	1.1690×10^{10}	1138s
SoSPD-Fusion	75.2%	1.1664×10^{10}	133s

Table 8.4: Evaluation of SoSPD: Numerical results for stereo reconstruction, for the two images in Figure 8.4.

	Energy @ 10s	Final energy	Time
FGBZ-Gradient	4.17×10^8	2.353×10^8	86s
HOCR-Gradient	4.35×10^8	2.368×10^8	78s
GRD-Gradient	6.72×10^8	2.348×10^8	776s
SoSPD-Gradient	2.87×10^8	2.347×10^8	42s

Table 8.5: Evaluation of SoSPD: Numerical results for denoising, averaged over the 100 images in the test set. For the second column, we stop both methods after 10 seconds, and compare energy values.

publicly available code, so we are left with the class of fusion-reduction methods [18, 40, 43]. While the Generalized Roof Duality method of [43] can produce good solutions, it is typically much slower than [18, 40], and is restricted to cliques of size at most 4. We observed that it obtains similar or slightly-worse energy values to SoSPD, while taking at least 10x more time, even for the heuristic version of GRD. We therefore focus on FGBZ [18] and HOCR [40] due to their speed and generality.

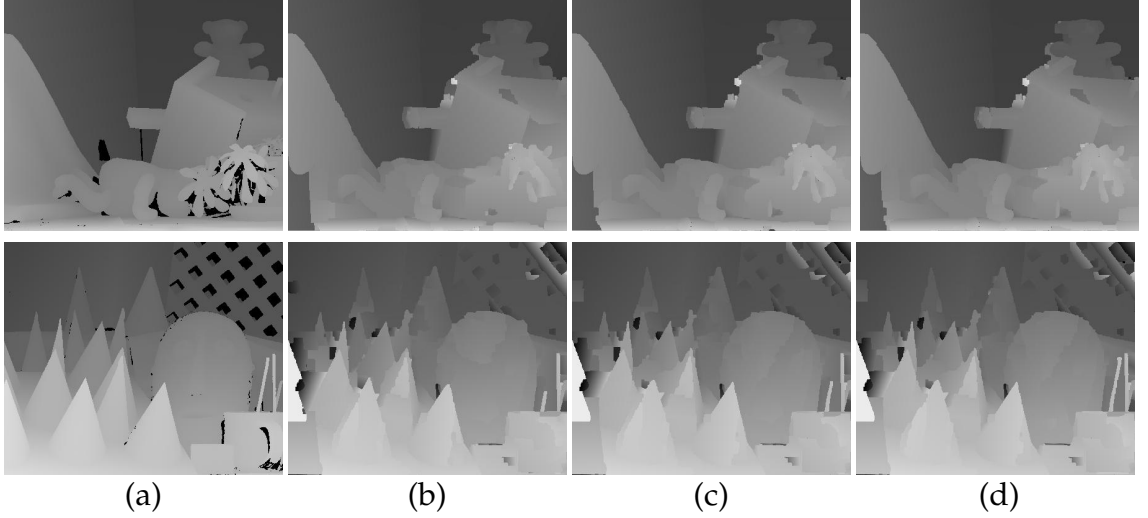


Figure 8.4: Evaluation of SoSPD: visual results for stereo reconstruction. (a) Ground truth disparities, with results from (b) FGBZ-Fusion (c) SoSPD-Fusion and (d) SoSPD-Best-Fusion. Top row is the “teddy” image, bottom row is “cones”. Results for SoSPD have slightly more correct pixels, and converged much faster — see Table 8.4 for details.

8.3.1 Stereo reconstruction

We have two variants of SoSPD for this experiment, which only differ in the choice of proposed moves. The first, SoSPD-Fusion, rotates through the 14 labels, and successively chooses each to be an α -expansion proposal for that iteration. The second, SoSPD-Best-Fusion, uses an idea from [5] to pick the best α for each iteration. More specifically, we choose the α which will have the greatest total capacity leaving the source, in order to encourage as many nodes to switch to lower height labels as possible. We compared with the baselines, FGBZ-Fusion using the reduction [18], and HOCR-Fusion using the reduction [40]. Both methods cycle through the pre-generated proposals and perform fusion move.

Numerical results are in Table 8.4 and images in Figure 8.4. Overall, the

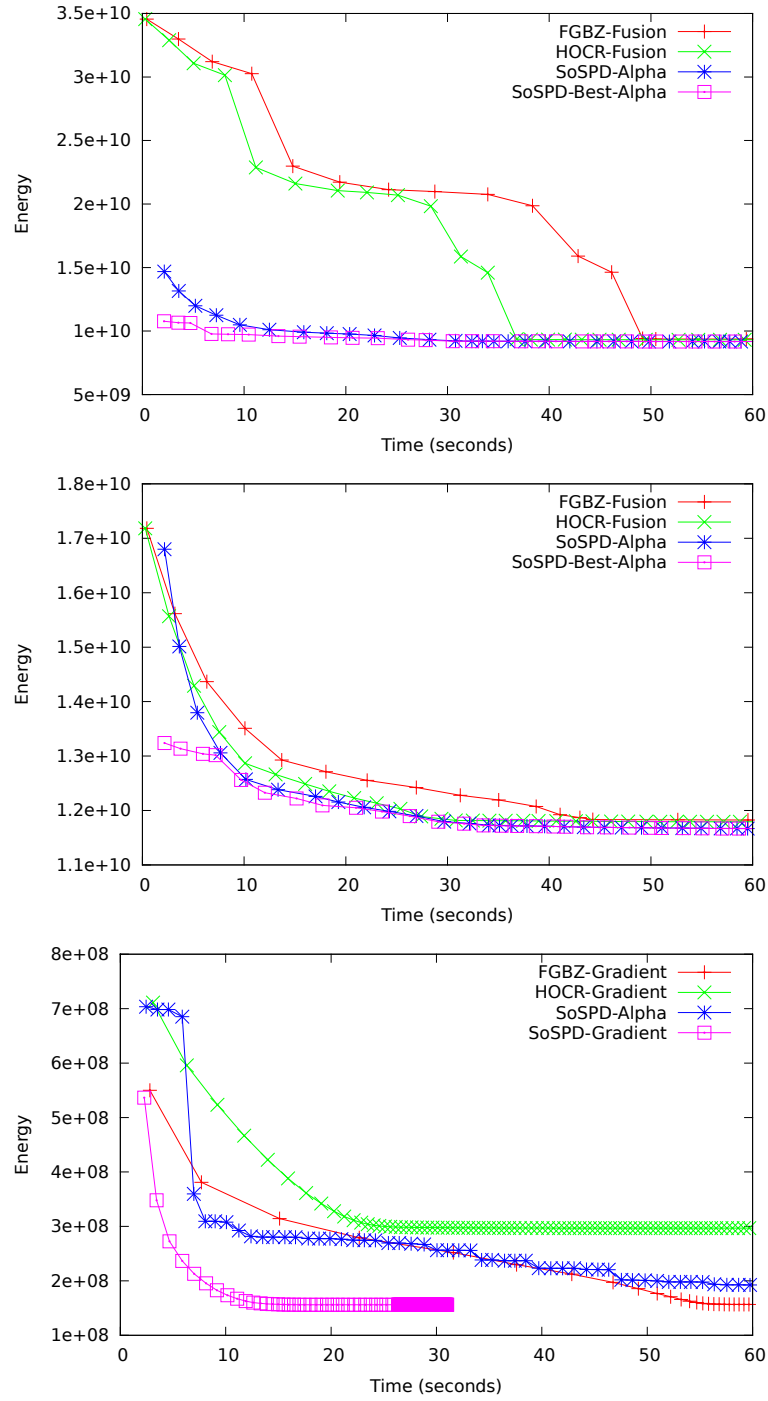


Figure 8.5: Evaluation of SoSPD: Energy reduction over time for the stereo images (top) “teddy” (center) “cones”. (bottom) Energy reduction over time for the denoising image “penguin”. Note that, in addition to converging faster, for a fixed time budget we achieve much better energy than the baseline.

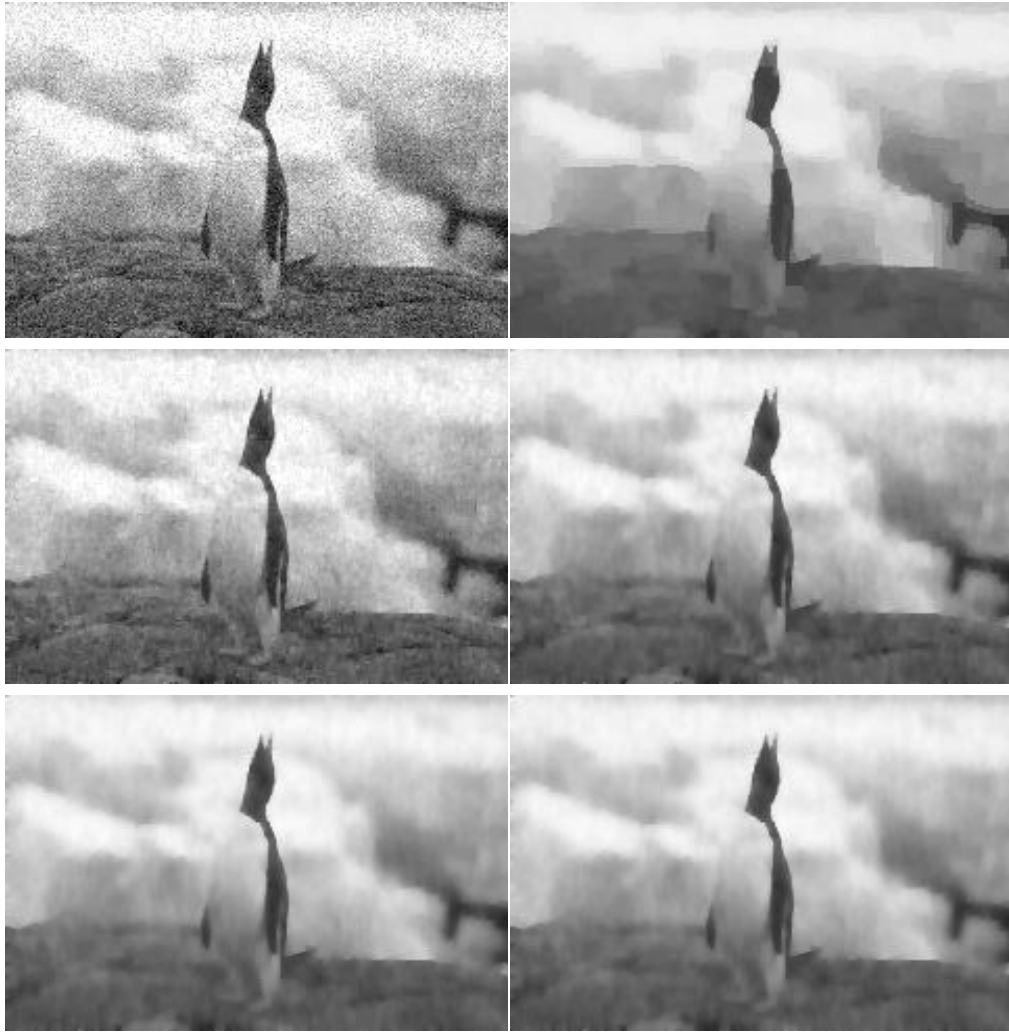


Figure 8.6: Evaluation of SoSPD: Visual results for Field of Experts denoising. (top left) noisy image (top right) SoSPD- α (center left) FGBZ-Gradient, 10 sec (center right) SoSPD-Gradient, 10 sec (bottom left) FGBZ-Gradient at convergence (bottom right) SoSPD-Gradient at convergence.

SoSPD variants and reduction methods reach similar energy and visual results; however, SoSPD is fastest overall (2.5x-3.5x vs FGBZ, 1.3x-1.5x vs HOCR).

8.3.2 Field of Experts denoising

SoSPD with α -expansion decreases the energy quickly initially, but gets stuck in poor local optima, with flat images as seen in Figure 8.6. Fortunately, gradient descent proposals [38] have been shown to be very effective at optimizing FoE priors. We call the combination of SoSPD with these fusion proposals SoSPD-Gradient.

We compare against fusion move with the same proposals, and the reductions of [18] and [40]. Overall, when comparing SoSPD vs. [18] for the same proposal method, SoSPD is significantly faster, and achieves slightly lower energy at convergence. Additionally, given a fixed time budget of 10 seconds, both the energy and visual results of SoSPD are significantly better, as seen in Figure 8.6 and Table 8.5.

CHAPTER 9

STRUCTURED LEARNING OF SUM-OF-SUBMODULAR HIGHER ORDER ENERGY FUNCTIONS

Now that we've covered inference of MRFs, we will turn to the second question of optimization: modeling. SoS functions can naturally express higher order priors involving, e.g., local image patches; however, it is difficult to fully exploit their expressive power because they have so many parameters. Rather than trying to formulate existing higher order priors as an SoS function, we take a discriminative learning approach, effectively searching the space of SoS functions for a higher order prior that performs well on our training set. We adopt a structural SVM approach [41, 95] and formulate the training problem in terms of quadratic programming; as a result we can efficiently search the space of SoS priors via an extended cutting-plane algorithm. We also show how the state-of-the-art max flow method for vision problems [30] can be modified to efficiently solve the submodular flow problem. Experimental comparisons are made against the OpenCV implementation of the GrabCut interactive segmentation technique [79], which uses hand-tuned parameters instead of machine learning. On a standard dataset [32] our method learns higher order priors with hundreds of parameter values, and produces significantly better segmentations. While our focus is on binary labeling problems, we show that our techniques can be naturally generalized to handle more than two labels.

9.1 Introduction

Discrete optimization methods such as graph cuts [12, 52] have proven to be quite effective for many computer vision problems, including stereo [12], inter-

active segmentation [79] and texture synthesis [61]. The underlying optimization problem behind graph cuts is a special case of submodular function optimization that can be solved exactly using max flow [52]. Graph cut methods, however, are limited by their reliance on first-order priors involving pairs of pixels, and there is considerable interest in expressing priors that rely on local image patches such as the popular Field of Experts model [77].

While SoS functions have more expressive power, they also involve a large number of parameters. Rather than addressing the question of which existing higher order priors can be expressed as an SoS function, we take a discriminative learning approach and effectively search the space of SoS functions with the goal of finding a higher order prior that gives strong results on our training set.¹

Our main contribution is to introduce the first learning method for training such SoS functions, and to demonstrate the effectiveness of this approach for interactive segmentation using learned higher order priors. Following a Structural SVM approach [41, 95], we show that the training problem can be cast as a quadratic optimization problem over an extended set of linear constraints. This generalizes large-margin training of pairwise submodular (a.k.a. regular [52]) MRFs [2, 91, 94], where submodularity corresponds to a simple non-negativity constraint. To solve the training problem, we show that an extended cutting-plane algorithm can efficiently search the space of SoS functions.

¹Since we are taking a discriminative approach, the higher-order energy function we learn does not have a natural probabilistic interpretation. We are using the word “prior” here somewhat loosely, as is common in computer vision papers that focus on energy minimization.

9.2 Related Work

Many learning problems in computer vision can be cast as structured output prediction, which allows learning outputs with spatial coherence. Among the most popular generic methods for structured output learning are Conditional Random Fields (CRFs) trained by maximum conditional likelihood [65], Maximum-Margin Markov Networks (M3N) [93], and Structural Support Vector Machines (SVM-struct) [95, 41]. A key advantage of M3N and SVM-struct over CRFs is that training does not require computation of the partition function. Among the two large-margin approaches M3N and SVM-struct, we follow the SVM-struct methodology since it allows the use of efficient inference procedures during training.

In this paper, we will learn submodular discriminant functions. Prior work on learning submodular functions falls into three categories: submodular function regression [4], maximization of submodular discriminant functions, and minimization of submodular discriminant functions.

Learning of submodular discriminant functions where a prediction is computed through maximization has widespread use in information retrieval, where submodularity models diversity in the ranking of a search engine [103, 67] or in an automatically generated abstract [87]. While exact (monotone) submodular maximization is intractible, approximate inference using a simple greedy algorithm has approximation guarantees and generally excellent performance in practice.

The models considered in this paper use submodular discriminant functions where a prediction is computed through minimization. The most popular such

models are regular MRFs [52]. Traditionally, the parameters of these models have been tuned by hand, but several learning methods exist. Most closely related to the work in this paper are Associative Markov Networks [94, 2], which take an M3N approach and exploit the fact that regular MRFs have an integral linear relaxation. These linear programs (LP) are folded into the M3N quadratic program (QP) that is then solved as a monolithic QP. In contrast, SVM-struct training using cutting planes for regular MRFs [91] allows graph cut inference also during training, and [17, 60] show that this approach has interesting approximation properties even the for multi-class case where graph cut inference is only approximate. More complex models for learning spatially coherent priors include separate training for unary and pairwise potentials [64], learning MRFs with functional gradient boosting [71], and the \mathcal{P}^n Potts models, all of which have had success on a variety of vision problems. Note that our general approach for learning multi-label SoS functions, described in section 9.3.4, includes the \mathcal{P}^n Potts model as a special case.

9.3 S3SVM: SoS Structured SVMs

In this section, we first review the SVM algorithm and its associated Quadratic Program (section 9.3.1). We then describe a general class of SoS discriminant functions which can be learned by SVM-struct (section 9.3.2) and explain this learning procedure (section 9.3.3). Finally, we generalize SoS functions to the multi-label case (section 9.3.4).

9.3.1 Structured SVMs

Structured output prediction describes the problem of learning a function $h : \mathcal{X} \rightarrow \mathcal{Y}$ where \mathcal{X} is the space of inputs, and \mathcal{Y} is the space of (multivariate and structured) outputs for a given problem. To learn h , we assume that a training sample of input-output pairs $S = ((x_1, y_1), \dots, (x_n, y_n)) \in (\mathcal{X} \times \mathcal{Y})^n$ is available and drawn i.i.d. from an unknown distribution. The goal is to find a function h from some hypothesis space \mathcal{H} that has low prediction error, relative to a loss function $\Delta(y, \bar{y})$. The function Δ quantifies the error associated with predicting \bar{y} when y is the correct output value. For example, for image segmentation, a natural loss function might be the Hamming distance between the true segmentation and the predicted labeling.

The mechanism by which Structural SVMs finds a hypothesis h is to learn a discriminant function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ over input/output pairs. One derives a prediction for a given input x by minimizing f over all $y \in \mathcal{Y}$.² We will write this as $h_{\mathbf{w}}(x) = \operatorname{argmin}_{y \in \mathcal{Y}} f_{\mathbf{w}}(x, y)$. We assume $f_{\mathbf{w}}(x, y)$ is linear in two quantities \mathbf{w} and Ψ $f_{\mathbf{w}}(x, y) = \mathbf{w}^T \Psi(x, y)$ where $\mathbf{w} \in \mathbb{R}^N$ is a parameter vector and $\Psi(x, y)$ is a feature vector relating input x and output y . Intuitively, one can think of $f_{\mathbf{w}}(x, y)$ as a cost function that measures how poorly the output y matches the given input x .

Ideally, we would find weights \mathbf{w} such that the hypothesis $h_{\mathbf{w}}$ always gives correct results on the training set. Stated another way, for each example x_i , the correct prediction y_i should have low discriminant value, while incorrect predictions \bar{y}_i with large loss should have high discriminant values. We write this

²Note that the use of minimization departs from the usual language of [95, 41] where the hypothesis is $\operatorname{argmax} f_{\mathbf{w}}(x, y)$. However, because of the prevalence of cost functions throughout computer vision, we have replaced f by $-f$ throughout.

constraint as a linear inequality in \mathbf{w}

$$\mathbf{w}^T \Psi(x_i, \bar{y}_i) \geq \mathbf{w}^T \Psi(x_i, y_i) + \Delta(y_i, \bar{y}_i) : \forall \bar{y} \in \mathcal{Y}. \quad (9.1)$$

It is convenient to define $\delta\Psi_i(\bar{y}) = \Psi(x_i, \bar{y}) - \Psi(x_i, y_i)$, so that the above inequality becomes $\mathbf{w}^T \delta\Psi_i(\bar{y}_i) \geq \Delta(y_i, \bar{y}_i)$.

Since it may not be possible to satisfy all these conditions exactly, we also add a slack variable to the constraint for each example i . Intuitively, the slack variable ξ_i represents the maximum misprediction loss on the i th example. Since we want to minimize the prediction error, we add an objective function which penalizes large slack. Finally, we also penalize $\|\mathbf{w}\|^2$ to discourage overfitting, with a regularization parameter C to trade off these costs.

Quadratic Program 1. n -SLACK STRUCTURAL SVM

$$\begin{aligned} \min_{\mathbf{w}, \xi \geq 0} & \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{n} \sum_{i=1}^n \xi_i \\ & \mathbf{w}^T \delta\Psi_i(\bar{y}_i) \geq \Delta(y_i, \bar{y}_i) - \xi_i \quad \forall i, \forall \bar{y}_i \in \mathcal{Y} \end{aligned}$$

9.3.2 Submodular Feature Encoding

We now apply the Structured SVM (SVM-struct) framework to the problem of learning SoS functions.

For the moment, assume our prediction task is to assign a binary label for each element of a base set V . We will cover the multi-label case in section 9.3.4. Since the labels are binary, prediction consists of assigning a subset $S \subseteq V$ for each input (namely the set S of pixels labeled 1).

Our goal is to construct a feature vector Ψ that, when used with the SVM-struct algorithm of section 9.3.1, will allow us to learn sum-of-submodular en-

ergy functions. Let's begin with the simplest case of learning a discriminant function $f_{C,\mathbf{w}}(S) = \mathbf{w}^T \Psi(S)$, defined only on a single clique and which does not depend on the input x .

Intuitively, our parameters \mathbf{w} will correspond to the table of values of the clique function f_C , and our feature vector Ψ will be chosen so that $\mathbf{w}_S = f_C(S)$. We can accomplish this by letting Ψ and \mathbf{w} have $2^{|C|}$ entries, indexed by subsets $T \subseteq C$, and defining $\Psi_T(S) = \delta_T(S)$ (where $\delta_T(S)$ is 1 if $S = T$ and 0 otherwise). Note that, as we claimed,

$$f_{C,\mathbf{w}}(S) = \mathbf{w}^T \Psi(S) = \sum_{T \subseteq C} \mathbf{w}_T \delta_T(S) = \mathbf{w}_S. \quad (9.2)$$

If our parameters \mathbf{w}_T are allowed to vary over all $\mathbb{R}^{2^{|C|}}$, then $f_C(S)$ may be an arbitrary function $2^C \rightarrow \mathbb{R}$, and not necessarily submodular. However, we can enforce submodularity by adding a number of linear inequalities. Recall that f is submodular if and only if $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$. Therefore, $f_{C,\mathbf{w}}$ is submodular if and only if the parameters satisfy

$$\mathbf{w}_{A \cup B} + \mathbf{w}_{A \cap B} \leq \mathbf{w}_A + \mathbf{w}_B : \forall A, B \subseteq C \quad (9.3)$$

These are just linear constraints in \mathbf{w} , so we can add them as additional constraints to Quadratic Program 1. There are $O(2^{|C|})$ of them, but each clique has $2^{|C|}$ parameters, so this does not increase the asymptotic size of the QP.

Theorem 88. *By choosing feature vector $\Psi_T(S) = \delta_T(S)$ and adding the linear constraints (9.3) to Quadratic Program 1, the learned discriminant function $f_w(S)$ is the maximum margin function f_C , where f_C is allowed to vary over all possible submodular functions $f : 2^C \rightarrow \mathbb{R}$.*

Proof. By adding constraints (9.3) to the QP, we ensure that the optimal solution \mathbf{w} defines a submodular $f_{\mathbf{w}}$. Conversely, for any submodular function f_C , there is a feasible \mathbf{w} defined by $\mathbf{w}_T = f_C(T)$, so the optimal solution to the QP must be the maximum-margin such function. \square

To introduce a dependence on the data x , we can define Ψ^{data} to be $\Psi_T^{\text{data}}(S, x) = \delta_T(S)\Phi(x)$ for an arbitrary nonnegative function $\Phi : \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$.

Corollary 89. *With feature vector Ψ^{data} and adding linear constraints (9.3) to QP 1, the learned discriminant function is the maximum margin function $f_C(S)\Phi(x)$, where f_C is allowed to vary over all possible submodular functions.*

Proof. Because $\Phi(x)$ is nonnegative, constraints (9.3) ensure that the discriminant function is again submodular. \square

Finally, we can learn multiple clique potentials simultaneously. If we have a neighborhood structure C with m cliques, each with a data-dependence $\Phi_C(x)$, we create a feature vector Ψ^{sos} composed of concatenating the m different features Ψ_C^{data} .

Corollary 90. *With feature vector Ψ^{sos} , and adding a copy of the constraints (9.3) for each clique C , the learned f_w is the maximum margin f of the form*

$$f(x, S) = \sum_{C \in C} f_C(S)\Phi_C(x) \quad (9.4)$$

where the f_C can vary over all possible submodular functions on the cliques C .

9.3.3 Solving the quadratic program

```

1: Input:  $S = ((x_1, y_1), \dots, (x_n, y_n)), C, \epsilon$ 
2:  $\mathcal{W} \leftarrow \emptyset$ 
3: repeat
4:   Recompute the QP solution with the current constraint set:
      $(\mathbf{w}, \xi) \leftarrow \operatorname{argmin}_{\mathbf{w}, \xi \geq 0} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C\xi$ 
     s.t. for all  $(\bar{y}_1, \dots, \bar{y}_n) \in \mathcal{W}$  :
        $\frac{1}{n} \mathbf{w}^T \sum_{i=1}^n \delta \Psi_i(\bar{y}_i) \geq \frac{1}{n} \sum_{i=1}^n \Delta(y_i, \bar{y}_i) - \xi$ 
     s.t. for all  $C \in \mathcal{C}, A, B \subseteq C$  :
        $\mathbf{w}_{C, A \cup B} + \mathbf{w}_{C, A \cap B} \leq \mathbf{w}_{C, A} + \mathbf{w}_{C, B}$ 
5:   for  $i=1, \dots, n$  do
6:     Compute the maximum violated constraint:
      $\hat{y}_i \leftarrow \operatorname{argmin}_{\hat{y} \in \mathcal{Y}} \{\mathbf{w}^T \Psi(x_i, \hat{y}) - \Delta(y_i, \hat{y})\}$ 
     by using IBFS to minimize  $f_{\mathbf{w}}(x_i, \hat{y}) - \Delta(y_i, \hat{y})$ .
7:   end for
8:    $\mathcal{W} \leftarrow \mathcal{W} \cup \{(\hat{y}_1, \dots, \hat{y}_n)\}$ 
9: until the slack of the max-violated constraint is  $\leq \xi + \epsilon$ .
10: return  $(\mathbf{w}, \xi)$ 

```

Algorithm 2: : S3SVM via the 1-Slack Formulation.

The n -slack formulation for SSVMs (QP 1) makes intuitive sense, from the point of view of minimizing the misprediction error on the training set. However, in practice it is better to use the 1-slack reformulation of this QP from [41]. Compared to n -slack, the 1-slack QP can be solved several orders of magnitude faster in practice, as well as having asymptotically better complexity.

The 1-slack formulation is an equivalent QP which replaces the n slack variables ξ_i with a single variable ξ . The loss constraints (9.1) are replaced with constraints penalizing the sum of losses across all training examples. We also include submodular constraints on \mathbf{w} .

Quadratic Program 2. 1-SLACK STRUCTURAL SVM

$$\begin{aligned}
& \min_{\mathbf{w}, \xi \geq 0} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \xi \\
& \text{s.t. } \frac{1}{n} \mathbf{w}^T \sum_{i=1}^n \delta \Psi_i(\bar{y}_i) \geq \frac{1}{n} \sum_{i=1}^n \Delta(y_i, \bar{y}_i) - \xi \quad \forall (\bar{y}_1, \dots, \bar{y}_n) \in \mathcal{Y}^n \\
& \mathbf{w}_{C, A \cup B} + \mathbf{w}_{C, A \cap B} \leq \mathbf{w}_{C, A} + \mathbf{w}_{C, B} \quad \forall C \in \mathcal{C}, A, B \subseteq C
\end{aligned} \tag{9.5}$$

Note that we have a constraint for each tuple $(\bar{y}_1, \dots, \bar{y}_n) \in \mathcal{Y}^n$, which is an exponential sized set. Despite the large set of constraints, we can solve this QP to any desired precision ϵ by using the cutting plane algorithm. This algorithm keeps track of a set \mathcal{W} of current constraints, and solves the current QP with regard to those constraints, and then given a solution (\mathbf{w}, ξ) , finds the most violated constraint and adds it to \mathcal{W} . Finding the most violated constraint consists of solving for each example x_i the problem

$$\hat{y}_i = \operatorname{argmin}_{\hat{y} \in \mathcal{Y}} f_{\mathbf{w}}(x, \hat{y}) - \Delta(y_i, \hat{y}). \tag{9.6}$$

Since the features Ψ ensure that $f_{\mathbf{w}}$ is SoS, then as long as Δ factors as a sum over the cliques C (for instance, the Hamming loss is such a function), then (9.6) can be solved with Submodular IBFS. Note that this also allows us to add arbitrary additional features for learning the unary potentials as well. Pseudocode for the entire S3SVM learning is given in Algorithm 2.

9.3.4 Generalization to multi-label prediction

Submodular functions are intrinsically binary functions. In order to handle the multi-label case, we use expansion moves [12] to reduce the multi-label optimization problem to a series of binary subproblems, where each pixel may ei-

ther switch to a given label α or keep its current label. If every binary subproblem of computing the optimal expansion move is an SoS problem, we will call the original multi-label energy function an SoS expansion energy.

Let L be our label set, with output space $\mathcal{Y} = L^V$. Our learned function will have the form $f(y) = \sum_{C \in \mathcal{C}} f_C(y_C)$ where $f_C : L^C \rightarrow \mathbb{R}$. For a clique C and label ℓ , define $C_\ell = \{i \mid y_i = \ell\}$, i.e., the subset of C taking label ℓ .

Theorem 91. *If all the clique functions are of the form*

$$f_C(y_C) = \sum_{\ell \in L} g_\ell(C_\ell) \quad (9.7)$$

where each g_ℓ is submodular, then any expansion move for the multi-label energy function f will be SoS.

Proof. Fix a current labeling y , and let $B(S)$ be the energy when the set S switches to label α . We can write $B(S)$ in terms of the clique functions and sets C_ℓ as

$$B(S) = \sum_{C \in \mathcal{C}} \left(g_\alpha(C_\alpha \cup S) + \sum_{\ell \neq \alpha} g_\ell(C_\ell \setminus S) \right) \quad (9.8)$$

We use a fact from the theory of submodular functions: if $f(S)$ is submodular, then for any fixed T both $f(T \cup S)$ and $f(T \setminus S)$ are also submodular. Therefore, $B(S)$ is SoS. \square

Theorem 91 characterizes a large class of SoS expansion energies. These functions generalize commonly used multi-label clique functions, including the \mathcal{P}^n Potts model [48]. The \mathcal{P}^n model pays cost λ_i when all pixels are equal to label i , and λ_{\max} otherwise. We can write this as an SoS expansion energy by letting $g_\ell(S) = \lambda_i - \lambda_{\max}$ if $S = C$ and otherwise 0. Then, $\sum_\ell g_\ell(S)$ is equal to the \mathcal{P}^n Potts model, up to an additive constant. Generalizations such as the robust \mathcal{P}^n

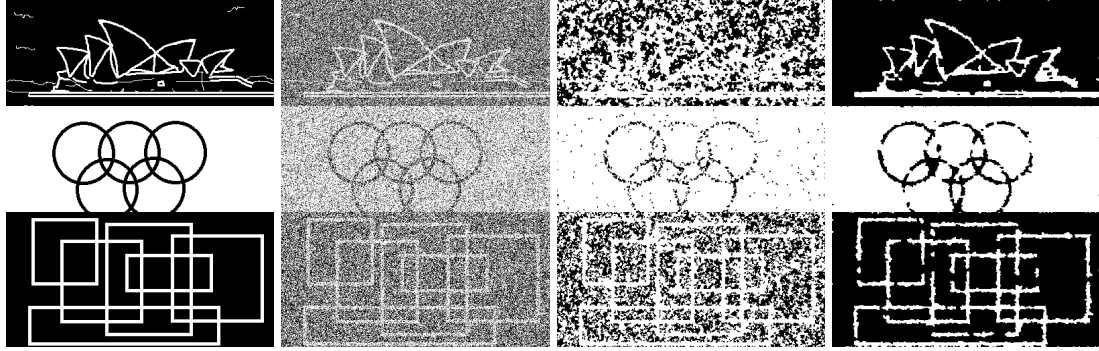


Figure 9.1: Example images from the binary segmentation results. From left to right, the columns are (a) the original image (b) the noisy input (c) results from Generic Cuts [3] (d) our results.

model [49] can be encoded in a similar fashion. Finally, in order to learn these functions, we let Ψ be composed of copies of Ψ^{data} — one for each g_ℓ , and add corresponding copies of the constraints (9.3).

As a final note: even though the individual expansion moves can be computed optimally, α -expansion still may not find the global optimum for the multi-labeled energy. However, in practice α -expansion finds good local optima, and has been used for inference in Structural SVM with good results, as in [60].

9.4 Experimental Results

In order to evaluate our algorithms, we focused on binary denoising and interactive segmentation. For binary denoising, Generic Cuts [3] provides the most natural comparison since it is a state-of-the-art method that uses SoS priors. For interactive segmentation the natural comparison is against GrabCut [79], where we used the OpenCV implementation. We ran our general S3SVM method,

which can learn an arbitrary SoS function, and also considered the special case of only using pairwise priors. For both the denoising and segmentation applications, we significantly improve on the accuracy of the hand-tuned energy functions.

9.4.1 Binary denoising

Our binary denoising dataset consists of a set of 20 black and white images. Each image is 400×200 and either a set of geometric lines, or a hand-drawn sketch (see Figure 9.1). We were unable to obtain the original data used by [3], so we created our own similar data by adding independent Gaussian noise at each pixel.

For denoising, the hand-tuned Generic Cuts algorithm of [3] posed a simple MRF, with unary pixels equal to the absolute valued distance from the noisy input, and an SoS prior, where each 2×2 clique penalizes the square-root of the number of edges with different labeled endpoints within that clique. There is a single parameter λ , which is the tradeoff between the unary energy and the smoothness term. The neighborhood structure C consists of all 2×2 patches of the image.

Our learned prior includes the same unary terms and clique structure, but instead of the square-root smoothness prior, we learn a clique function g to get an MRF $E_{\text{SVM}}(y) = \sum_i |y_i - x_i| + \sum_{C \in C} g(y_C)$. Note that each clique has the same energy as every other, so this is analogous to a graph cuts prior where each pairwise edge has the same attractive potential. Our energy function has 16 total parameters (one for each possible value of g , which is defined on 2×2

patches).

We randomly divided the 20 input images into 10 training images and 10 test images. The loss function was the Hamming distance between the correct, un-noisy image and the predicted image. To hand tune the value λ , we picked the value which gave the minimum pixel-wise error on the training set. S3SVM training took only 16 minutes.

Numerically, S3SVM performed significantly better than the hand-tuned method, with an average pixel-wise error of only 4.9% on the training set, compared to 28.6% for Generic Cuts. The time needed to do inference after training was similar for both methods: 0.82 sec/image for S3SVM vs. 0.76 sec/image for Generic Cuts. Visually, the S3SVM images are significantly cleaner looking, as shown in Figure 9.1.

9.4.2 Interactive segmentation

The input to interactive segmentation is a color image, together with a set of sparse foreground/background annotations provided by the user. See Figure 9.2 for examples. From the small set of labeled foreground and background pixels, the prediction task is to recover the ground-truth segmentation for the whole image.

Our baseline comparison is the Grabcut algorithm, which solves a pairwise CRF. The unary terms of the CRF are obtained by fitting a Gaussian Mixture Model to the histograms of pixels labeled as being definitely foreground or background. The pairwise terms are a standard contrast-sensitive Potts poten-

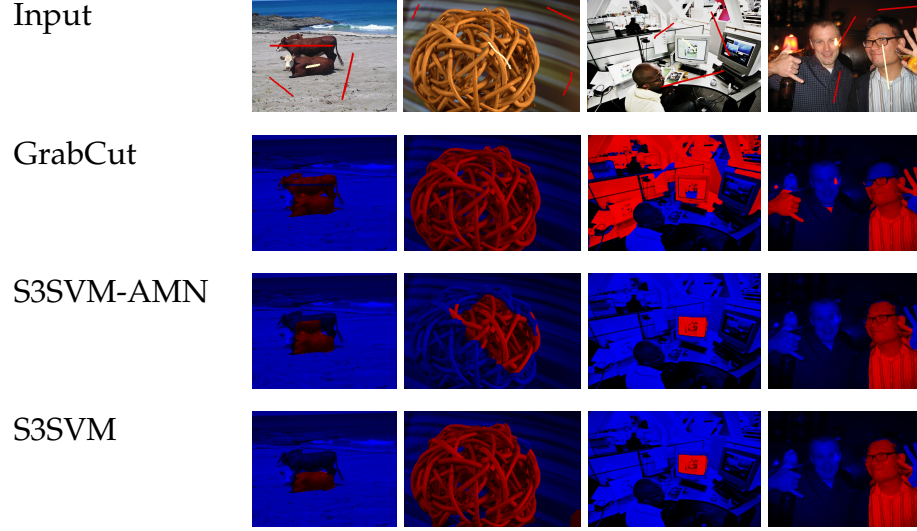


Figure 9.2: Example images from binary segmentation results. Input with user annotations are shown at top, with results below.

tial, where the cost of pixels i and j taking different labels is equal to $\lambda \cdot \exp(-\beta |x_i - x_j|)$ for some hand-coded parameters β, λ . Our primary comparison is against the OpenCV implementation of Grabcut, available at www.opencv.org.

As a special case, our algorithm can be applied to pairwise-submodular energy functions, for which it solves the same optimization problem as in Associative Markov Networks (AMN's) [94, 2]. Automatically learning parameters allows us to add a large number of learned unary features to the CRF.

As a result, in addition to the smoothness parameter λ , we also learn the relative weights of approximately 400 features describing the color values near a pixel, and relative distances to the nearest labeled foreground/background pixel. Further details on these features can be found in the Supplementary Material. We refer to this method as S3SVM-AMN.

Our general S3SVM method can incorporate higher-order priors instead of

just pairwise ones. In addition to the unary features used in S3SVM-AMN, we add a sum-of-submodular higher-order CRF. Each 2×2 patch in the image has a learned submodular clique function. To obtain the benefits of the contrast-sensitive pairwise potentials for the higher-order case, we cluster (using k -means) the x and y gradient responses of each patch into 50 clusters, and learn one submodular potential for each cluster. Note that S3SVM automatically allows learning the entire energy function, including the clique potentials and unary potentials (which come from the data) simultaneously.

We use a standard interactive segmentation dataset from [32] of 151 images with annotations, together with pixel-level segmentations provided as ground truth. These images were randomly sorted into training, validation and testing sets, of size 75, 38 and 38 respectively. We trained both S3SVM-AMN and S3SVM on the training set for various values of the regularization parameter c , and picked the value c which gave the best accuracy on the validation set, and report the results of that value c on the test set.

The overall performance is shown in the table below. Training time is measured in seconds, and testing time in seconds per image. Our implementation, which used the submodular flow algorithm based on IBFS discussed in section 5.2, will be made freely available under the MIT license.

Algorithm	Average error	Training	Testing
Grabcut	$10.6 \pm 1.4\%$	n/a	1.44
S3SVM-AMN	$7.5 \pm 0.5\%$	29000	0.99
S3SVM	$7.3 \pm 0.5\%$	92000	1.67

Learning and validation was performed 5 times with independently sam-

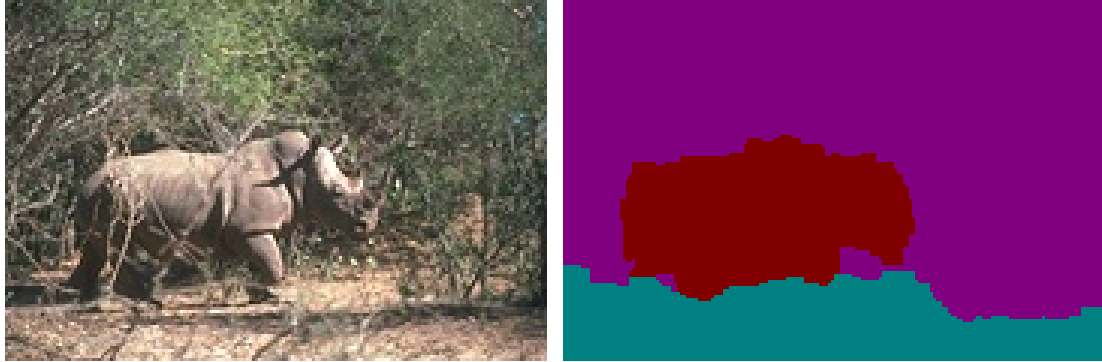


Figure 9.3: A multi-label segmentation result, on data from [36]. The purple label represents vegetation, red is rhino/hippo and blue is ground. There are 7 labels in the input problem, though only 3 are present in the output we obtain on this particular image.

pled training sets. The averages and standard deviations shown above are from these 5 samples.

While our focus is on binary labeling problems, we have conducted some preliminary experiments with the multi-label version of our method described in section 9.3.4. A sample result is shown in figure 9.3, using an image taken the Corel dataset used in [36].

CHAPTER 10

CONCLUSION

In this thesis, we set out to expand the set of models for which fast inference is possible. As we’ve noted in the introduction, modeling and inference are tightly coupled — applications demand more accurate solutions which require more sophisticated models, however they are limited to the kinds of problems for which fast inference algorithms are known.

For computer vision problems, Markov Random Fields continue to be the tool of choice for encoding spatial relations between pixels in an image. We’ve shown that many useful properties of images cannot be encoded by first-order MRFs, and that many natural features of images, especially their local, patch-based statistics, are best encoded by higher-order MRFs.

When developing optimization algorithms for higher-order MRFs, we have been able to leverage existing graph-cuts methods, using reduction methods to turn higher-order MRFs into first-order problems. We’ve also seen that the basic ideas behind graph-cuts can be generalized to higher-order models, including alpha-expansion and other primal large-neighborhood search algorithms.

For designing higher-order versions of graph-cuts methods, the key concept appears to be submodularity. This has been known to be the necessary condition for first-order MRFs since [52]; however, the condition for first-order graphs is much simpler than the general case. In particular, we have found that Sum-of-Submodular inference is a natural middle-ground between min-cut based inference and fully general submodular function minimization with

$$\text{MIN-CUT} \subseteq \text{SOS MINIMIZATION} \subseteq \text{SUBMODULAR MINIMIZATION} \quad (10.1)$$

In particular, we are able to take advantage of the clique structure by treating the cliques as a hypergraph over the variables — this allows a fairly straightforward generalization of augmenting paths based algorithms to the higher order case (including the state-of-the-art for vision inputs, IBFS [30]). In this algorithm, we noted that submodularity is the key property for augmenting paths to find a globally optimal solution.

For multilabel problems, our key tool has been Linear Programming, and in particular, the Local Marginal Polytope relaxation of the MRF inference problem. The primary feature of linear programming based algorithms is that they can actually say something about the global behavior of the problem, and in particular, using duality we get a global lower bound on the optimal solution. This is in contrast with many popular primal-only algorithms such as alpha-expansion and fusion-moves which make local choices (even if they are searching over a very large local neighborhood), and which cannot say anything about the global optimum.

Furthermore, we’ve seen that the LP dual is also useful to speed up inference algorithms, by guiding the binary subproblems within a fusion-move algorithm, as done by the primal-dual algorithm SoSPD. In particular, we have generalized the FastPD algorithm [59] for first-order MRFs to work on higher-order problems, with many of the same speedups over pure-primal algorithms. Furthermore, we’ve used the dual LP to prove approximation ratios for our algorithm, giving a guaranteed bound on how far we can be from the optimum solution.

A major limitation of higher-order MRFs is that they are difficult to design by hand, as they have many more parameters than first-order models. We have

explored one method for learning higher-order models, using a Structural SVM approach; however, many more learning algorithms are possible. A key feature of this (and related) learning algorithms, though, is that they require repeated application of inference. As a result, every time new inference algorithms allow new models to be efficiently optimized, we can do learning on these models as well.

Finally, we will note that higher-order MRFs are a heavyweight solution for many applications. The algorithms presented in this thesis have brought higher-order MRFs from being largely-intractable to being reasonably fast to optimize. However, the algorithms involved still scale poorly with the clique size (typically $O(2^{|C|})$) and generally require minutes to run, compared to the milliseconds required for real-time performance. However, for achieving maximum accuracy, they present a much greater flexibility for modeling and encoding of constraints.

APPENDIX A

LOCAL COMPLETENESS

We are considering a general labeling problem, with variables x_1, \dots, x_n , taking labels in sets L_1, \dots, L_n . In an MRF the energy function can be written as a sum of clique energies: there is some set C of cliques, and clique functions f_C such that

$$E(x_1, \dots, x_n) = \sum_C f_C(\mathbf{x}_C) \quad (\text{A.1})$$

To minimize this energy with respect to a fusion move, we have an input image I and a proposed image I' , and for each pixel a binary variable b_k that encodes whether the k -th pixel takes label I_k or I'_k .

The energy function is now a sum of clique energies over these n binary variables. For a clique C of size d , we can write the clique energy f_C as a sum of terms in the binary variables and their negations, by specifying the energy pointwise for each possible assignment of the d binary variables in C :

- For each assignment $\gamma \in \mathbb{B}^d$, where $\mathbb{B} = \{0, 1\}$, let $f(\gamma)$ be the energy of the clique in the fused image according to γ . For instance, with $d = 4$ and $\gamma = (0, 1, 1, 0)$ we have $f(\gamma) = f_C(I_0, I'_1, I'_2, I_3)$.
- Let $b^{(\gamma)}$ be the term whose i -th literal is b_i or \bar{b}_i , according to whether γ_i is 1 or 0 respectively. For $\gamma = (0, 1, 1, 0)$, we would have $b^{(\gamma)} = \bar{b}_0 b_1 b_2 \bar{b}_3$.
- Note that the term $b^{(\gamma)}$ is 1 exactly when the binary variables (b_1, \dots, b_d) take the assignment γ . Thus, we can write the clique energy as:

$$f_C(b_1, \dots, b_d) = \sum_{\gamma \in \mathbb{B}^d} f(\gamma) b^{(\gamma)} \quad (\text{A.2})$$

The first step of our reduction is to transform this to a multilinear polynomial by substituting $1 - b_i$ for $\overline{b_i}$ each time a negated variable occurs. This could possibly result in terms with coefficients for each subset of b_1, \dots, b_d .

For each subset $S \subseteq \{b_1, \dots, b_d\}$, we can actually calculate the coefficient on the term $t_S = \prod_{j \in S} b_j$. Let Γ_S be the set of assignments $\gamma \in \mathbb{B}^d$ with $\gamma_i = 0$ for $i \notin S$, and let

$$\sigma(\gamma) = \begin{cases} 1 & \text{The number of 0s in } \gamma \text{ is even} \\ -1 & \text{otherwise} \end{cases} \quad (\text{A.3})$$

Then, after we substitute $(1 - b_i)$ for all occurrences of $\overline{b_i}$ and collect all terms with the same variables, the coefficient on the term t_S is

$$\text{Coeff}(t_S) = \sum_{\gamma \in \Gamma_S} \sigma(\gamma) f(\gamma) \quad (\text{A.4})$$

Therefore, to show that our energy function is locally dense, it suffices to show that $\text{Coeff}(t_S)$ is never (or rarely) 0 for any subset $S \subsetneq \{b_1, \dots, b_d\}$ (note that we don't care if $t_{\{x_1, \dots, x_d\}}$ has coefficient 0, since it is not a subset of any term).

We can obtain a general theorem about the binary energy functions corresponding to fusion moves, by moving to a continuous framework. We embed the original intensities in \mathbb{R} , and extend the clique energies f_C to functions on \mathbb{R}^d . We need two assumptions: (1) f_C is $d - 1$ times continuously differentiable and (2) each of the d different mixed partials $\frac{\partial^{d-1} f}{\partial x_1 \dots \widehat{\partial x_i} \dots \partial x_d}$ (where $\widehat{\partial x_i}$ means to omit the i -th partial) take their zeros in a set of measure 0.

Theorem 92. *Under these two assumptions the set of proposed-current image pairs (I, I') for which the fusion move binary energy function does not have local density 1 has measure 0 as a subset of $\mathbb{R}^n \times \mathbb{R}^n$.*

Proof. By the above argument, it suffices to show that for a clique C on variables x_1, \dots, x_d , the current and proposed images (I_C, I'_C) which have $\text{Coeff}(t_S) = 0$ for each $S \subseteq \{x_1, \dots, x_d\}$ have measure 0 in $\mathbb{R}^d \times \mathbb{R}^d$.

For every fusion move (I_C, I'_C) and assignment $\gamma \in \mathbb{B}^d$, we get a point $v^{(\gamma)}$ in \mathbb{R}^d , the result of fusion on just the clique pixels: $v_k^{(\gamma)} = \begin{cases} I_k & \gamma_k = 0 \\ I'_k & \gamma_k = 1 \end{cases}$

If we list out these points, we get the set

$$(I_1, \dots, I_{d-1}, I_d), (I_1, \dots, I_{d-1}, I'_d), (I_1, \dots, I'_{d-1}, I_d), \dots, (I'_1, \dots, I'_{d-1}, I_d), (I'_1, \dots, I'_d)$$

containing each possible fusion of I_C and I'_C . Note that these 2^d points form an axis-aligned rectangular prism in \mathbb{R}^d . Denote these points as $\text{Verts}(I_C, I'_C)$. Every fusion move (I_C, I'_C) gives a rectangular prism in this fashion.

Now, fix S , and let the bad set of fusion-moves, B , be those (I_C, I'_C) for which $\text{Coeff}(t_S) = 0$. To produce a contradiction, assume that B has nonzero measure. It is a closed set, so it contains an open ball. So there is some fusion move (x_0, y_0) and radius δ such that for all $\epsilon_x, \epsilon_y \in \mathbb{R}^d$ with $|\epsilon_x|, |\epsilon_y| < \delta$, the fusion-move $(x_0 + \epsilon_x, y_0 + \epsilon_y)$ is still a bad fusion-move.

Now, since $\text{Coeff}(t_S) = 0$ for the fusion move (x_0, y_0) , we can manipulate equation A.4 to get that

$$f(v^{(1,1,\dots,1)}) = - \sum_{\gamma \in \Gamma_S \setminus \{(1,\dots,1)\}} \sigma(\gamma) f(v^{(\gamma)}) \quad (\text{A.5})$$

Since there's an open ball of bad fusion moves around (x_0, y_0) , for $\epsilon \in \mathbb{R}^d$ with $|\epsilon| < \delta$, the fusion move $(x_0, y_0 + \epsilon)$ is also bad. If we set $\epsilon^{(\gamma)}$ equal to $(\gamma_1 \epsilon_1, \dots, \gamma_n \epsilon_n)$ (i.e. the vector which is ϵ_i when γ_i is 1, and 0 otherwise), then the fusion move

$(x_0, y_0 + \epsilon)$ gives a rectangular prism with vertices $v^{(\gamma)} + \epsilon^{(\gamma)}$ for $\gamma \in \mathbb{B}^d$. Then, since $(x_0, y_0 + \epsilon)$ is still a bad fusion move, we can again manipulate equation A.4 to get

$$f(v^{(1,1,\dots,1)} + \epsilon) = - \sum_{\gamma \in \Gamma_S \setminus \{(1,\dots,1)\}} \sigma(\gamma) f(v^{(\gamma)} + \epsilon^{(\gamma)}) \quad (\text{A.6})$$

Let $g_\gamma(\epsilon) = f(v^{(\gamma)} + \epsilon^{(\gamma)})$. Notice that since $\epsilon_i^{(\gamma)} = 0$ whenever $\gamma_i = 0$, we have that this function only depends on the variables x_i where $\gamma_i = 1$. Thus, we have that for $\gamma \in \Gamma_S \setminus \{(1, \dots, 1)\}$, the function g_γ depends on at most $d - 2$ of the ϵ_i . This is because S is a proper subset of $\{x_1, \dots, x_d\}$, so all the γ have $\gamma_d = 0$, and then we remove the element $(1, \dots, 1)$ which has $d - 1$ 1s.

Therefore, we have that the mixed partial $\frac{\partial^{d-1} g_\gamma}{\partial x_1 \dots \partial x_{d-1}}$ is 0 for all $|\epsilon| < \delta$. Therefore, since $f(v^{(1,\dots,1)} + \epsilon)$ is a linear combination of the g_γ , it also has this mixed partial equal to 0. But then, we have found a set of nonzero measure (the open ball of radius δ around y_0) with mixed partial 0, contradicting our hypothesis.

Therefore, the set of bad fusion moves in fact must have measure 0.

□

APPENDIX B

LAPLACIAN EQUATIONS

In the proof of Lemma 74 in the main paper, in order to show the solution of Laplacian equation $M\psi = \Delta$ is non-negative, we claimed it's straightforward to show that the inverse of the coefficient matrix M^{-1} is nonnegative (meaning each component is nonnegative), hence $\psi = M^{-1}\Delta$ is non-negative (since Δ is also non-negative). Now, we will show M^{-1} is non-negative for the completeness.

It's useful to have the following fact about the inverse of a tridiagonal matrix [96]:

Lemma 93. *The inverse of a non-singular tridiagonal matrix M*

$$M = \begin{pmatrix} a_1 & b_1 & & & 0 \\ c_1 & a_2 & b_2 & & \\ & c_2 & \ddots & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ 0 & & & c_{n-1} & a_n \end{pmatrix} \quad (\text{B.1})$$

is given by

$$(M^{-1})_{ij} = \begin{cases} (-1)^{i+j} \prod_{k=i}^{j-1} b_k \theta_{i-1} \phi_{j+1} / \theta_n, & \text{if } i \leq j \\ (-1)^{i+j} \prod_{k=j}^{i-1} c_k \theta_{j-1} \phi_{i+1} / \theta_n, & \text{if } i > j \end{cases} \quad (\text{B.2})$$

where

$$\begin{aligned} \theta_i &= a_i \theta_{i-1} - b_{i-1} c_{i-1} \theta_{i-2} & \text{for } i = 2, 3, \dots, n \\ \phi_i &= a_i \phi_{i+1} - b_i c_i \phi_{i+2} & \text{for } i = n-1, \dots, 1 \end{aligned} \quad (\text{B.3})$$

with initial values $\theta_0 = 1, \theta_1 = a_1, \phi_{n+1} = 1, \phi_n = a_n$.

In our Laplacian equations, we have $a_i = 2, \forall i$ and $b_j = c_j = -1, \forall j$. Substitute

them into Lemma 93, we have:

$$(M^{-1})_{ij} = \begin{cases} \theta_{i-1}\phi_{j+1}/\theta_n, & \text{if } i \leq j \\ \theta_{j-1}\phi_{i+1}/\theta_n, & \text{if } i > j \end{cases} \quad (\text{B.4})$$

where

$$\begin{aligned} \theta_i &= 2\theta_{i-1} - \theta_{i-2} & \text{for } i = 2, 3, \dots, n \\ \phi_i &= 2\phi_{i+1} - \phi_{i+2} & \text{for } i = n-1, \dots, 1 \end{aligned} \quad (\text{B.5})$$

with initial values $\theta_0 = \phi_{n+1} = 1, \theta_1 = \phi_n = 2$.

It's easy to use induction to show $\theta_i = i + 1$ and $\phi_i = n + 2 - i$ from their recursive definition in (B.5). Therefore, we have

$$(M^{-1})_{ij} = \begin{cases} \frac{i \cdot (n + 1 - j)}{n + 1}, & \text{if } i \leq j \\ \frac{j \cdot (n + 1 - i)}{n + 1}, & \text{if } i > j \end{cases} \quad (\text{B.6})$$

Clearly, we have M^{-1} to be a positive matrix in our Laplacian equations.

APPENDIX C

APPROXIMATION RATIO FOR CARDINALITY UPPER BOUNDS

Theorem 79. *The cardinality-based upper bound gives a $2^{(1-\frac{1}{p})}$ -approximation.*

Proof. Similar to the proof of Theorem 76, we can rewrite the objective for odd n as

$$\|\psi\|_p := \left(\sum_{k=0}^n \binom{n}{k} \psi_k^p \right)^{\frac{1}{p}} = \left(\sum_{k=0}^{(n-1)/2} \binom{n}{k} (\psi_k^p + \psi_{n-k}^p) \right)^{\frac{1}{p}} \quad (\text{C.1})$$

and for even n :

$$\begin{aligned} \|\psi\|_p &= \left(\sum_{k=0}^{n/2-1} \binom{n}{k} (\psi_k^p + \psi_{n-k}^p) \right. \\ &\quad \left. + \frac{1}{2} \binom{n}{n/2} (\psi_{n/2}^p + \psi_{n/2}^p) \right)^{\frac{1}{p}} \end{aligned} \quad (\text{C.2})$$

Let's use $\Psi_k = \psi_k^p + \psi_{n-k}^p$ as a shorthand. We can see the objective can be represented as

$$\|\psi\|_p := \left(\sum_{k=0}^{\lceil \frac{n}{2} \rceil} a_k \Psi_k \right)^{\frac{1}{p}} \quad (\text{C.3})$$

with $a_k \geq 0, \forall k$.

Consider ψ^* and $\bar{\psi}$ as the true optimal solution and our approximation solution. Define Ψ^* and $\bar{\Psi}$ accordingly. Consider each term Ψ_k , we must have $\Psi_k^* \geq \frac{L_k^p}{2^{p-1}}$ since the RHS is the solution for the following program:

$$\begin{aligned} &\min_{\psi} \Psi_k \\ &\text{s.t. } \Psi_k = \psi_k^p + \psi_{n-k}^p \\ &\quad \psi_k + \psi_{n-k} \geq L_k, \\ &\quad \psi_k, \psi_{n-k} \geq 0 \end{aligned} \quad (\text{C.4})$$

where the minimizer is achieved by $\psi_k = \psi_{n-k} = \frac{L_k}{2}$.¹

¹Recall L_k is the lower bound of $\psi_k + \psi_{n-k}$ which only depends on Δ .

Meanwhile, we must have $\bar{\Psi} \leq L_k^p$ since the RHS is the solution for the following program:

$$\begin{aligned}
& \max_{\psi} \Psi_k \\
& \text{s.t. } \Psi_k = \psi_k^p + \psi_{n-k}^p \\
& \psi_k + \psi_{n-k} = L_k, \\
& \psi_k, \psi_{n-k} \geq 0
\end{aligned} \tag{C.5}$$

where the maximizer is achieved by either $\psi_k = 0, \psi_{n-k} = L_k$ or $\psi_k = L_k, \psi_{n-k} = 0$.²

Therefore, we must have $\frac{\bar{\Psi}_k}{\Psi_k^*} \leq 2^{p-1}$ for $\forall k$. As a non-negative linear combination of non-negative numbers, we also have

$$\frac{\sum_{k=0}^{\lceil \frac{n}{2} \rceil} a_k \bar{\Psi}_k}{\sum_{k=0}^{\lceil \frac{n}{2} \rceil} a_k \Psi_k^*} \leq 2^{p-1} \tag{C.6}$$

hence

$$\frac{\left(\sum_{k=0}^{\lceil \frac{n}{2} \rceil} a_k \bar{\Psi}_k \right)^{\frac{1}{p}}}{\left(\sum_{k=0}^{\lceil \frac{n}{2} \rceil} a_k \Psi_k^* \right)^{\frac{1}{p}}} \leq 2^{(1-\frac{1}{p})} \tag{C.7}$$

□

²Recall we proved in Lemma 5 in the main paper that our approximation $\bar{\psi}$ must let each $\bar{\Psi}_k = \bar{\psi}_k + \bar{\psi}_{n-k}$ achieves its lower bound L_k , hence we have the second constraint.

BIBLIOGRAPHY

- [1] Björn Andres, Jörg H. Kappes, Ullrich Köthe, Christoph Schnörr, and Fred A. Hamprecht. An empirical comparison of inference algorithms for graphical models with higher order factors using opengm. In *DAGM-Symposium*, pages 353–362, 2010.
- [2] Dragomir Anguelov, Benjamin Taskar, Vassil Chatalbashev, Daphne Koller, Dinkar Gupta, Jeremy Heitz, and Andrew Y. Ng. Discriminative learning of Markov Random Fields for segmentation of 3D scan data. In *CVPR*, pages 169–176, 2005.
- [3] Chetan Arora, Subhashis Banerjee, Prem Kalra, and S. N. Maheshwari. Generic cuts: an efficient algorithm for optimal inference in higher order MRF-MAP. In *ECCV*, 2012.
- [4] Maria-Florina Balcan and Nicholas J. A. Harvey. Learning submodular functions. In *ACM Symposium on Theory of Computing (STOC)*, pages 793–802, 2011.
- [5] D. Batra and P. Kohli. Making the right moves: Guiding alpha-expansion using local primal-dual gaps. In *CVPR*, pages 1865–1872, 2011.
- [6] I. Ben Ayed, L. Gorelick, and Y. Boykov. Auxiliary cuts for general classes of higher order functionals. In *CVPR*, 2013.
- [7] J. Besag. On the statistical analysis of dirty pictures (with discussion). *Journal of the Royal Statistical Society, Series B*, 48(3):259–302, 1986.
- [8] E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1-3), 2002.
- [9] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- [10] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *TPAMI*, 26(9):1124–1137, 2004.
- [11] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *International Conference on Computer Vision (ICCV)*, pages 377–384, 1999.

- [12] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *TPAMI*, 23(11):1222–1239, 2001.
- [13] Qifeng Chen and Vladlen Koltun. Fast MRF optimization with application to depth reconstruction. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 3914–3921, 2014.
- [14] B. V. Cherkassky and A. V. Goldberg. On implementing push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.
- [15] J. Edmonds and R. Giles. A min-max relation for submodular functions on graphs. *Annals of Discrete Mathematics*, 1:185–204, 1977.
- [16] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In Michael Jnger, Gerhard Reinelt, and Giovanni Rinaldi, editors, *Combinatorial Optimization Eureka, You Shrink!*, volume 2570 of *Lecture Notes in Computer Science*, pages 11–26. Springer Berlin Heidelberg, 2003.
- [17] T. Finley and T. Joachims. Training structural SVMs when exact inference is intractable. In *International Conference on Machine Learning (ICML)*, pages 304–311, 2008.
- [18] A. Fix, A. Gruber, E. Boros, and R. Zabih. A graph cut algorithm for higher-order Markov Random Fields. In *ICCV*, 2011.
- [19] A. Fix, T. Joachims, S. Park, and R. Zabih. Structured learning of sum-of-submodular higher order energy functions. In *ICCV*, 2013.
- [20] Alexander Fix and Sameer Agarwal. *Duality and the Continuous Graphical Model*, pages 266–281. Springer International Publishing, Cham, 2014.
- [21] Alexander Fix, Joyce Chen, Endre Boros, and Ramin Zabih. *Computer Vision – ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part I*, chapter Approximate MRF Inference Using Bounded Treewidth Subgraphs, pages 385–398. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [22] Alexander Fix, Chen Wang, and Ramin Zabih. A primal-dual algorithm for higher-order multilabel markov random fields. In *CVPR, 2014*. Supplemental Material at www.cs.cornell.edu/~afix/.

- [23] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
- [24] D. Freedman and P. Drineas. Energy minimization via graph cuts: Settling what is possible. In *CVPR*, 2005.
- [25] Brendan Frey and David MacKay. A revolution: Belief propagation in graphs with cycles. In *Neural Information Processing Systems (NIPS)*, 1997.
- [26] S Fujishige and X Zhang. A push/relabel framework for submodular flows and its refinement for 0-1 submodular flows. *Optimization*, 38(2):133–154, 1996.
- [27] Andrew C Gallagher, Dhruv Batra, and Devi Parikh. Inference for order reduction in markov random fields. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1857–1864. IEEE, 2011.
- [28] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *TPAMI*, 6:721–741, 1984.
- [29] Amir Globerson and Tommi S. Jaakkola. Fixing max-product: Convergent message passing algorithms for map lp-relaxations. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 553–560. Curran Associates, Inc., 2008.
- [30] Andrew V. Goldberg, Sagi Hed, Haim Kaplan, Robert E. Tarjan, and Renato F. Werneck. Maximum flows by incremental breadth-first search. In *European Symposium on Algorithms*, pages 457–468. Springer-Verlag, 2011.
- [31] Lena Gorelick, Yuri Boykov, Olga Veksler, Ismail Ben Ayed, and Andrew Delong. Submodularization for binary pairwise energies. In *CVPR*, 2014.
- [32] V. Gulshan, C. Rother, A. Criminisi, A. Blake, and A. Zisserman. Geodesic star convexity for interactive image segmentation. In *CVPR*, 2010.
- [33] P. L. Hammer, P. Hansen, and B. Simeone. Roof duality, complementation and persistency in quadratic 0-1 optimization. *Mathematical Programming*, 28:121–155, 1984.
- [34] P.L. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas*. Springer, 1968.

- [35] J. M. Hammersley and P. E. Clifford. Markov random fields on finite graphs and lattices. Unpublished manuscript, 1971.
- [36] Xuming He, Richard S. Zemel, and Miguel Á. Carreira-Perpiñán. Multi-scale conditional random fields for image labeling. In *CVPR*, pages 695–703, 2004.
- [37] H. Ishikawa. Higher-order clique reduction in binary graph cut. In *CVPR*, 2009.
- [38] H. Ishikawa. Higher-order gradient descent by fusion-move graph cut. In *ICCV*, 2009.
- [39] Hiroshi Ishikawa. Exact optimization for Markov Random Fields with convex priors. *TPAMI*, 25(10):1333–1336, 2003.
- [40] Hiroshi Ishikawa. Transformation of general binary MRF minimization to the first order case. *TPAMI*, 33(6), 2010.
- [41] T. Joachims, T. Finley, and Chun-Nam Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.
- [42] Vladimir Jojic, Stephen Gould, and Daphne Koller. Accelerated dual decomposition for map inference. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 503–510, 2010.
- [43] F. Kahl and P. Strandmark. Generalized roof duality for pseudo-boolean optimization. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 255–262, 2011.
- [44] Fredrik Kahl and Petter Strandmark. Generalized roof duality. *Discrete Applied Mathematics*, 160(1617):2419 – 2434, 2012.
- [45] Jorg H Kappes, Bjoern Andres, Fred A Hamprecht, Christoph Schnorr, Sebastian Nowozin, Dhruv Batra, Sungwoong Kim, Bernhard X Kausler, Jan Lellmann, Nikos Komodakis, et al. A comparative study of modern inference techniques for discrete energy minimization problems. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1328–1335. IEEE, 2013.
- [46] B. M. Kelm, N. Mueller, B. H. Menze, and F. A. Hamprecht. Bayesian estimation of smooth parameter maps for dynamic contrast-enhanced mr

images with block-icm. In *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW '06. Conference on*, pages 96–96, June 2006.

- [47] Jon Kleinberg and Eva Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and Markov Random Fields. *J. ACM*, 49(5):616–639, 2002. ACM Press.
- [48] Pushmeet Kohli, M. Pawan Kumar, and Philip H.S. Torr. P3 and beyond: Move making algorithms for solving higher order functions. *TPAMI*, 31(9):1645–1656, 2008.
- [49] Pushmeet Kohli, Lubor Ladick, and Philip Torr. Robust higher order potentials for enforcing label consistency. *IJCV*, 82:302–324, 2009.
- [50] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. In *International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2005.
- [51] V. Kolmogorov and C. Rother. Minimizing nonsubmodular functions with graph cuts-a review. *TPAMI*, 29(7):1274–1279, July 2007. Earlier version appears as technical report MSR-TR-2006-100.
- [52] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *TPAMI*, 26(2):147–59, 2004.
- [53] Vladimir Kolmogorov. Minimizing a sum of submodular functions. *Discrete Appl. Math.*, 160(15):2246–2258, October 2012.
- [54] Vladimir Kolmogorov and Thomas Schoenemann. Generalized sequential tree-reweighted message passing. *CoRR*, abs/1205.6352, 2012.
- [55] N. Komodakis and N. Paragios. Beyond pairwise energies: Efficient optimization for higher-order MRFs. In *CVPR*, pages 2985–2992, 2009.
- [56] Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. Mrf energy minimization and beyond via dual decomposition. In *IN: IEEE PAMI.*, 2011.
- [57] Nikos Komodakis and Georgios Tziritas. A new framework for approximate labeling via graph cuts. In *International Conference on Computer Vision (ICCV)*, 2005.

- [58] Nikos Komodakis and Georgios Tziritas. Approximate labeling via graph cuts based on linear programming. *TPAMI*, 29(8):1436–1453, 2007.
- [59] Nikos Komodakis, Georgios Tziritas, and Nikos Paragios. Fast primal-dual strategies for MRF optimization. Technical Report 0605, Ecole Centrale de Paris, 2006.
- [60] H. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic labeling of 3D point clouds for indoor scenes. In *Conference on Neural Information Processing Systems (NIPS)*, 2011.
- [61] Vivek Kwatra, Arno Schodl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *SIGGRAPH*, 2003.
- [62] Dongjin Kwon, Kyong Joon Lee, Il Dong Yun, and Sang Uk Lee. Non-rigid image registration using dynamic higher-order mrf model. In *ECCV*, pages 373–386, 2008.
- [63] Lubor Ladicky, Chris Russell, Pushmeet Kohli, and Philip H. S. Torr. *Computer Vision – ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part V*, chapter Graph Cut Based Inference with Co-occurrence Statistics, pages 239–253. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [64] Lubor Ladicky, Christopher Russell, Pushmeet Kohli, and Philip H. S. Torr. Associative hierarchical CRFs for object class image segmentation. In *ICCV*, pages 739–746, 2009.
- [65] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.
- [66] V Lempitsky, C Rother, S Roth, and A Blake. Fusion moves for Markov Random Field optimization. *TPAMI*, 32(8):1392–1405, Aug 2010.
- [67] Hui Lin and Jeff Bilmes. Learning mixtures of submodular shells with application to document summarization. In *UAI*, pages 479–490, 2012.
- [68] L. Lovász. *Submodular functions and convexity*, pages 235–257. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

- [69] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, September 1994.
- [70] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, 2001.
- [71] D. Munoz, J. A. Bagnell, N. Vandapel, and M. Hebert. Contextual classification with functional max-margin Markov networks. In *CVPR*, pages 975–982, 2009.
- [72] Claudia Nieuwenhuis, Eno Ttpe, Lena Gorelick, Olga Veksler, and Yuri Boykov. Efficient regularization of squared curvature. *CoRR*, abs/1311.1838, 2013.
- [73] James B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Math. Program.*, 118(2):237–251, January 2009.
- [74] Judeah Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.
- [75] R. Potts. Some generalized order-disorder transformations. *Proceedings of the Cambridge Philosophical Society*, 48:106–109, 1952.
- [76] I.G. Rosenberg. Reduction of bivalent maximization to the quadratic case. Technical report, Centre d’Etudes de Recherche Oprationnelle, 1975.
- [77] Stefan Roth and Michael Black. Fields of experts. *IJCV*, 82:205–229, 2009.
- [78] C. Rother, P. Kohli, W. Feng, and J.Y. Jia. Minimizing sparse higher order energy functions of discrete variables. In *CVPR*, pages 1382–1389, 2009.
- [79] C. Rother, V. Kolmogorov, and A. Blake. “GrabCut” - interactive foreground extraction using iterated graph cuts. *SIGGRAPH*, 23(3):309–314, 2004.
- [80] C. Rother, S. Kumar, V. Kolmogorov, and A. Blake. Digital tapestry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
- [81] Bogdan Savchynskyy, Stefan Schmidt, Jörg Kappes, and Christoph Schnörr. Efficient mrf energy minimization via adaptive diminishing

- smoothing. *Uncertainty in Artificial Intelligence, UAI-2012*, pages 746–755, 2012.
- [82] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.
 - [83] Daniel Scharstein and Richard Szeliski. High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–195. IEEE, 2003.
 - [84] Dmitriy Schlesinger. Exact solution of permuted submodular minsum problems. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 28–38. 2007.
 - [85] Alexander Shekhovtsov, Pushmeet Kohli, and Carsten Rother. *Pattern Recognition: Joint 34th DAGM and 36th OAGM Symposium, Graz, Austria, August 28-31, 2012. Proceedings*, chapter Curvature Prior for MRF-Based Segmentation and Shape Inpainting, pages 41–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
 - [86] M.I. Shlezinger. Syntactic analysis of two-dimensional visual signals in the presence of noise. *Cybernetics*, 12(4):612–628, 1976.
 - [87] R. Sipos, P. Shivaswamy, and T. Joachims. Large-margin learning of submodular summarization models. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2012.
 - [88] Petter Strandmark and Fredrik Kahl. *Energy Minimization Methods in Computer Vision and Pattern Recognition: 8th International Conference, EMM-CVPR 2011, St. Petersburg, Russia, July 25-27, 2011. Proceedings*, chapter Curvature Regularization for Curves and Surfaces in a Global Optimization Framework, pages 205–218. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
 - [89] Paul Swoboda, Bogdan Savchynskyy, Jorg H. Kappes, and Christoph Schnorr. Partial optimality by pruning for map-inference with general graphical models. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.
 - [90] Rick Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A

- comparative study of energy minimization methods for Markov Random Fields. *TPAMI*, 30(6):1068–1080, 2008.
- [91] M. Szummer, P. Kohli, and D. Hoiem. Learning CRFs using graph cuts. In *ECCV*, pages II: 582–595, 2008.
 - [92] Meng Tang, Ismail Ben Ayed, and Yuri Boykov. Pseudo-bound optimization for binary energies. In *ECCV*, 2014.
 - [93] B. Taskar, C. Guestrin, and D. Koller. Maximum-margin markov networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.
 - [94] Benjamin Taskar, Vassil Chatalbashev, and Daphne Koller. Learning associative markov networks. In *ICML*, 2004.
 - [95] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *International Conference on Machine Learning (ICML)*, pages 104–112, 2004.
 - [96] Riaz A Usmani. Inversion of a tridiagonal jacobi matrix. *Linear Algebra and Its Applications*, 212:413–414, 1994.
 - [97] Chaohui Wang, Nikos Komodakis, and Nikos Paragios. Markov random field modeling, inference & learning in computer vision & image understanding: A survey. *Computer Vision and Image Understanding*, 117(11):1610 – 1627, 2013.
 - [98] Chaohui Wang, Olivier Teboul, Fabrice Michel, Salma Essafi, and Nikos Paragios. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2010: 13th International Conference, Beijing, China, September 20-24, 2010, Proceedings, Part III*, chapter 3D Knowledge-Based Segmentation Using Pose-Invariant Higher-Order Graphs, pages 189–196. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
 - [99] Y. Weiss, C. Yanover, and T. Meltzer. Map estimation, linear programming and belief propagation with convex free energies. In *Uncertainty in AI*, 2007.
 - [100] T. Werner. A linear programming approach to max-sum problem: A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(7):1165–1179, July 2007.

- [101] Tomáš Werner. High-arity interactions, polyhedral relaxations, and cutting plane algorithm for soft constraint optimisation (MAP-MRF). In *CVPR 2008: Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 109–116, Madison, USA, June 2008. IEEE Computer Society, Omnipress.
- [102] Oliver Woodford, Philip Torr, Ian Reid, and Andrew Fitzgibbon. Global stereo reconstruction under second-order smoothness priors. *TPAMI*, 31:2115–2128, 2009.
- [103] Yisong Yue and T. Joachims. Predicting diverse subsets using structural SVMs. In *International Conference on Machine Learning (ICML)*, pages 271–278, 2008.
- [104] Stanislav Zivny, David Cohen, and Peter Jeavons. The expressive power of binary submodular functions. In *Mathematical Foundations of Computer Science 2009*, volume 5734 of *LNCS*, pages 744–757. 2009.